DOCUMENT RESUME

ED 050 760 LI 002 820

TITLE	Feature Analysis of Generalized Data Base Management Systems.
INSTITUTION	Conference on Data Systems Languages, Monroeville, Pa. Systems Committee.
PUB DATE	May 71
NOTE	511p.
AVAILABLE FROM	Association for Computing Machinery Headquarters, Order Department, 1133 Avenue of the Americas, New York, N. Y. 10026 ($8.00)
EDRS PRICE	EDRS Price MF-$0.65 HC-$19.74
DESCRIPTORS	Computer Programs, Computers, *Data Bases, Information Retrieval, *Information Storage, *Information Systems, *Management Information Systems, *Management Systems, Operations Research, *Systems Analysis, Systems Development

ABSTRACT

A more complete definition of the features offered in present day generalized data base management systems is provided by this second technical report of the CODASYL Systems Committee. In a tutorial format, each feature description is followed by either narrative information covering ten systems or by a table for all systems. The ten systems covered in this report are: (1) the Conference on Data Systems Languages' COBOL as defined in the "Journal of Development," (2) the Data Base Task Group's proposal, (3) International Business Machines Corporation's (IBM's) GIS, (4) General Electric Company's (GE's) IDS, (5) IMB's IMS, (6) Informatics' MARK IV, (7) IBM's FFS (also known as NIPS), (8) Auerbach and Western Electric's SC-1, (9) Systems Development Corporation's +DMS and (10) Radio Corporation of America's UL/1. (MM)

U.S. DEPARTMENT OF HEALTH, EDUCATION
& WELFARE
OFFICE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL OFFICE OF EDUCATION POSITION OR POLICY

ED050760

CODASYL SYSTEMS COMMITTEE
TECHNICAL REPORT

FEATURE ANALYSIS OF GENERALIZED DATA BASE MANAGEMENT SYSTEMS

MAY 1971

1

ED050760

FEATURE ANALYSIS OF GENERALIZED DATA BASE MANAGEMENT SYSTEMS

by

CODASYL SYSTEMS COMMITTEE

MAY 1971

Chairman:

T. William Olle	RCA Corporation

Members:

Gordon C. Everest	University of Minnesota (and Auerbach Corporation)
James P. Fry	University of Michigan (formerly with MITRE Corporation)
Mary E. Fuller	URS Data Sciences Company
*Mary K. Hawes	Information Systems Leasing Corp.
*Anthony J. Kay	Honeywell Information Systems
Henry C. Lefkovits	Honeywell Information Systems (formerly with General Electric)
William C. McGee	IBM Corporation
A. Metaxides	Bell Telephone Laboratories
Ronald M. Olson	Control Data Corporation
*Martin Rich	Esso Mathematics and Systems
Richard F. Schubert	B. F. Goodrich Chemical
Edgar H. Sibley	University of Michigan
William H. Stieger	Chase Brass and Copper Company
Alfred H. Vorhaus	MITRE Corporation (formerly with SDC)
Arla E. Weinert	Naval Command Systems Support Activity
John W. Young	National Cash Register

*Resigned during 1970.

This report is prepared by the CODASYL Systems Committee and represents the viewpoint of its members, but not necessarily of their respective affiliations. The report is authorized by the CODASYL Executive Committee as a Systems Committee technical report for distribution outside CODASYL.

Copyright:

In accordance with CODASYL policy, this report may be reproduced or translated in whole or in part. However, it is requested that a mention be made of the source. If a major portion of the report is used it is requested that mention be made of the authors and of their affiliations in the above form.

LI 002 820

ADDITIONAL COPIES AVAILABLE
for $8.00 prepaid from:

Association for Computing Machinery
Order Department
1133 Avenue of the Americas
New York, NY 10026

In Europe Inquiries Should be Addressed to:

British Computer Society
29 Portland Place
London W1, England

or:

IFIP Data Processing Group
Stadhouder Skada 6
Amsterdam, Netherlands

ACKNOWLEDGEMENT

The work in preparing this report was performed by the members of the CODASYL Systems Committee whose names are listed on the front page. These individuals were supported by their affiliations, who made members' time available and who sponsored all travel and meeting expenses.

In addition to the members of the committee, a number of other individuals assisted the committee by reviewing drafts and answering questions on specific points. Appreciation is expressed by the Committee to the following for their assistance.

William B. Helgeson	Honeywell Information Systems Inc.
Ronald McDowell	Chevrolet Engineering Center
Martin J. Rich	ESSO Mathematics & Systems, Inc.
Joseph E. Sciulli & colleagues	Western Electric
Robert W. Taylor	University of Michigan
John Thurlow	ESSO Mathematics & Systems, Inc.
Thomas Work	International Business Machines

DISCLAIMER

The Systems Committee has made every attempt to ensure the accuracy of the information contained in the systems descriptions. Readers are requested to refer to the originator's source documentation to which reference is given in chapter 1. The Systems Committee disclaims responsibility for any inadvertent accuracies or misinterpretations.

INFORMATION

Information regarding the further activities of the CODASYL Systems Committee can be obtained from the following:

Chairman, CODASYL Systems Committee
P.O. Box 124
Monroeville, Pennsylvania 15146

TABLE OF CONTENTS

Page

INTRODUCTION.. I- 1

I.1 Background.. I- 1
I.2 Systems Committee goals.. I- 1
I.3 Decisions leading to preparation of current report........ I- 2
I.4 Functions of current report.. I- 3
I.5 Current state of the art .... I- 3
  I.5.1 Host language capabilities.. I- 5
  I.5.2 Self-contained capabilities.. I- 6
  I.5.3 Inter-system capabilities.. I- 7
  I.5.4 Data independence and binding.. I- 8
  I.5.5 User interface.. I- 8
  I.5.6 Levels of user.. I- 9

I.6 Technical problems facing designers.. I- 9
  I.6.1 Existing storage structures.. I-10
  I.6.2 More complex data structures.. I-10
  I.6.3 Self-contained operations on network structures.... I-11
  I.6.4 Non-programmer interaction.. I-11
  I.6.5 Unification of both approaches.. I-12

I.7 Use of COBOL as a basis for further development........... I-12
  I.7.1 Data structure capabilities.. I-13
  I.7.2 Host language capabilities.. I-14
  I.7.3 Self-contained capabilities.. I-14

I.8 Features of generalized data base management systems...... I-15
  I.8.1 General summary.. I-16
  I.8.2 Data structure.. I-16
  I.8.3 Data definition.. I-16
  I.8.4 Interrogation.. I-16
  I.8.5 Update.. I-17
  I.8.6 Creation.. I-17
  I.8.7 Programmer facilities.. I-18
  I.8.8 Data administrator functions.. I-18
  I.8.9 Storage structure.. I-18
  I.8.10 Operational environment.. I-19

I.9 Report conventions.. I-19
  I.9.1 Terminology.. I-19
  I.9.2 Language syntax specifications.. I-19
  I.9.3 Abbreviations.. I-20

TABLE OF CONTENTS (continued)

Page

I.10 Systems Committee role in future development............ I-20

References.. I-21

1. GENERAL SUMMARY.. 1- 1

1.1 Identification.. 1- 1
  1.1.1 Self-contained.................................... 1- 1
  1.1.2 Host language..................................... 1- 2

1.2 Data structure class.................................... 1- 4
1.3 Generalized processes provided.......................... 1- 6
1.4 Language type... 1- 7
1.5 Storage structure class................................. 1-10
1.6 System environment...................................... 1-11
1.7 System summaries.. 1-13
  1.7.1 Underlying philosophy............................. 1-13
  1.7.2 System descriptions............................... 1-17

1.8 Selected bibliography................................... 1-29

2. DATA STRUCTURES.. 2- 1

2.1 Items... 2- 5
  2.1.1 Item types.. 2- 5
    2.1.1.1 Numeric item types............................ 2- 7
    2.1.1.2 String item types............................. 2- 9
    2.1.1.3 Other item types.............................. 2- 9

  2.1.2 Item schema attributes............................ 2-12
    2.1.2.1 Names... 2-12
    2.1.2.2 Value class attributes........................ 2-14
    2.1.2.3 Other item schema attributes.................. 2-16

  2.1.3 Other item attributes............................. 2-20

2.2 Groups.. 2-22
  2.2.1 Group types....................................... 2-27
  2.2.2 Group composition................................. 2-31
    2.2.2.1 Schema composition............................ 2-31
    2.2.2.2 Instance composition.......................... 2-33

  2.2.3 Group attributes.................................. 2-34
    2.2.3.1 Names... 2-34
    2.2.3.2 Other schema attributes....................... 2-34
    2.2.3.3 Non-schema attributes......................... 2-39

TABLE OF CONTENTS (continued)

Page

2.3 Group relations 2-40
  2.3.1 Group relation types 2-47
  2.3.2 Group relation composition 2-49
    2.3.2.1 Schema composition 2-49
    2.3.2.2 Instance composition 2-50

  2.3.3 Group relation attributes 2-50
    2.3.3.1 Names 2-50
    2.3.3.2 Other schema attributes 2-51
    2.3.3.3 Non-schema attributes 2-52

2.4 Entries 2-52
  2.4.1 Entry types 2-59
  2.4.2 Entry composition 2-59
    2.4.2.1 Schema composition 2-59
    2.4.2.2 Instance composition 2-62

  2.4.3 Entry attributes 2-64
    2.4.3.1 Names 2-64
    2.4.3.2 Other schema attributes 2-64
    2.4.3.3 Non-schema attributes 2-65

2.5 Files 2-66
  2.5.1 File types 2-67
  2.5.2 File composition 2-67
    2.5.2.1 Schema composition 2-67
    2.5.2.2 Instance composition 2-73

  2.5.3 File attributes 2-74
    2.5.3.1 Names 2-74
    2.5.3.2 Other schema attributes 2-74
    2.5.3.3 Non-schema attributes 2-76

2.6 Data base 2-77
  2.6.1 Data base composition 2-77

2.7 Data structure generalization 2-78

3. DATA DEFINITION 3- 1

3.1 Context of the data definition 3- 1
3.2 Item schema definition 3- 5
3.3 Group schema definition 3-17
3.4 Group relation schema definition 3-26
3.5 Entry schema definition 3-30
3.6 File schema definition 3-33
3.7 Data base schema definition 3-41

TABLE OF CONTENTS (continued)

Page

3.8 Processing and storage of the data definition............. 3-41
3.9 Revision of the data definition........................... 3-44
3.10 Auxiliary data structure definition...................... 3-48
3.11 Sample data definitions................................... 3-58

4. INTERROGATION FUNCTION....................................... 4- 1

4.1 Characteristics of premise action relationship............ 4- 5
  4.1.1 Procedural language actions........................ 4- 7
  4.1.2 Non-procedural language actions.................... 4- 8

4.2 Identifying applicability of interrogation................ 4-10
4.3 Conditional expressions................................... 4-11
  4.3.1 Simple conditions.................................. 4-12
    4.3.1.1 Basic relational operators.................. 4-13
    4.3.1.2 Permitted form of subject in simple relational conditions................... 4-14
    4.3.1.3 Restrictions on the use of relational operators............................ 4-14
    4.3.1.4 Permitted reference quantities in simple relational conditions................... 4-15
    4.3.1.5 String matching conditions.................. 4-17
    4.3.1.6 Other relational operators.................. 4-18
    4.3.1.7 Existence conditions........................ 4-19

  4.3.2 Compound conditions................................ 4-20
    4.3.2.1 Logical connectors.......................... 4-21
    4.3.2.2 Precedence rule for logical connectors...... 4-22
    4.3.2.3 Levels of nesting........................... 4-22
    4.3.2.4 Compound conditions on the same item....... 4-23
    4.3.2.5 Logically connected conditions on different items................................. 4-25
    4.3.2.6 Logically connected conditions on several items but with same reference quantity..... 4-26

4.4 Conditions on groups...................................... 4-27
4.5 Identification of conditional expression.................. 4-30
4.6 Data extraction... 4-31
  4.6.1 Generalized extraction features.................... 4-32
  4.6.2 Multiple outputs per conditional expression........ 4-33
  4.6.3 Limit of output volume............................. 4-34
  4.6.4 Sorting.. 4-36

4.7 Item level extraction..................................... 4-38
  4.7.1 Item schema attributes............................. 4-38
  4.7.2 Item instance attributes........................... 4-39
  4.7.3 Item value decoding................................ 4-40
  4.7.4 Item level derived data............................ 4-41

TABLE OF CONTENTS (continued)

Page

4.8 Group level extraction.......... 4-42
  4.8.1 Group level derived data.......... 4-44

4.9 Entry level extraction.......... 4-44
  4.9.1 Entry level derived data.......... 4-45

4.10 File level extraction.......... 4-46
  4.10.1 File level derived data.......... 4-46
  4.10.2 Cross entry counting.......... 4-49

4.11 Report formatting facilities.......... 4-51
  4.11.1 Item level editing.......... 4-53
  4.11.2 Report body formatting.......... 4-54
    4.11.2.1 Item level placement.......... 4-54
    4.11.2.2 Report blocks.......... 4-56
    4.11.2.3 Page size control.......... 4-58

  4.11.3 Titles.......... 4-60
  4.11.4 Heading lines and footing lines.......... 4-62
  4.11.5 Other embedded literals.......... 4-64
  4.11.6 Control break facilities.......... 4-65

4.12 Interrogating the stored data definition.......... 4-67
4.13 Mechanized files.......... 4-68
  4.13.1 Files for system's use.......... 4-69
  4.13.2 Files for use outside system.......... 4-70

5. UPDATE.......... 5- 1

5.1 User update control.......... 5- 2
5.2 Data description.......... 5- 6
  5.2.1 Transaction definition.......... 5- 6
  5.2.2 Files used.......... 5-10

5.3 Transaction program definition.......... 5-12
  5.3.1 Transaction program format.......... 5-12
  5.3.2 Data mapping.......... 5-14
  5.3.3 User control over data access.......... 5-15
  5.3.4 Update data selection.......... 5-18
  5.3.5 Data base changes.......... 5-21
    5.3.5.1 Entry level changes.......... 5-25
    5.3.5.2 Group level changes.......... 5-28
    5.3.5.3 Item level changes.......... 5-30

  5.3.6 Transaction validation.......... 5-34
  5.3.7 Transaction editing and transformation.......... 5-36
  5.3.8 Other user specified features.......... 5-38

TABLE OF CONTENTS (continued)

Page

5.4 Auxiliary update functions.............................. 5-39
    5.4.1 Maintenance of item attributes..................... 5-39
    5.4.2 Maintenance of file storage........................ 5-40
    5.4.3 Ordering of transactions and files................. 5-41
    5.4.4 Maintenance of system integrity.................... 5-42

6. CREATION FUNCTIONS... 6- 1

6.1 Creation action cycle...................................... 6- 3
6.2 Definition of data and storage structure of input files.... 6- 7
6.3 Allocation of media space.................................. 6- 9
6.4 Provision of the input data file........................... 6-11
6.5 Population of the file..................................... 6-13
    6.5.1 Entry, group and item validation................... 6-16
    6.5.2 Transformations on data............................ 6-18

6.6 Monitoring creation functions.............................. 6-20

7. PROGRAMMING FACILITIES..................................... 7- 1

7.1 Summary of data manipulation language statements........... 7- 2
7.2 Modes of processing.. 7- 4
    7.2.1 Input, output, and update modes.................... 7- 4
    7.2.2 Random and sequential modes........................ 7- 5

7.3 Method of interface.. 7- 8
    7.3.1 Invocation of facilities from the host language.... 7- 8
    7.3.2 Language form...................................... 7-11
    7.3.3 Addressable data structures........................ 7-12

7.4 Program-system communication............................... 7-14
    7.4.1 Currency... 7-16
    7.4.2 User working area.................................. 7-20
    7.4.3 Error and exception conditions..................... 7-25
    7.4.4 Selection criteria................................. 7-30
    7.4.5 Security clearance and integrity................... 7-34

7.5 Data manipulation language statements...................... 7-38
    7.5.1 Control statements................................. 7-38
        7.5.1.1 Open... 7-38
        7.5.1.2 Close.. 7-42
        7.5.1.3 Conditional.................................. 7-44

    7.5.2 Data retrieval..................................... 7-45
        7.5.2.1 Locate....................................... 7-45
        7.5.2.2 Locate and access............................ 7-51
        7.5.2.3 Simple access................................ 7-55
        7.5.2.4 Hold and reprocess........................... 7-57
        7.5.2.5 Currency reset............................... 7-58

TABLE OF CONTENTS (continued)

Page

7.5.3 Data modification statements........................ 7-61
    7.5.3.1 Add...................................... 7-61
    7.5.3.2 Change................................... 7-65
    7.5.3.3 Delete................................... 7-68
    7.5.3.4 Reorder.................................. 7-71
    7.5.3.5 Reorganize............................... 7-74

7.5.4 Special purpose statements......................... 7-74
    7.5.4.1 Table handling........................... 7-75
    7.5.4.2 Communications........................... 7-77

7.6 Facilities for system programmers.................... 7-79
  7.6.1 Multiple levels of interface..................... 7-79
  7.6.2 Accessing the stored data definition............. 7-80
  7.6.3 Handling a generalized auxiliary data definition.... 7-81

8. DATA ADMINISTRATION FUNCTIONS......................... 8- 1

8.1 Systems administrator functions...................... 8- 4
8.2 Data administrator initiated process affecting other users. 8- 5
  8.2.1 Assignment of pass keys.......................... 8- 6
  8.2.2 Specifying the logging of modifications to the data base.......... 8- 7
  8.2.3 Specifying the logging of transactions........... 8- 9
  8.2.4 Specifying an audit trail......................... 8-10
  8.2.5 Storage of programs acting on the data base....... 8-11
  8.2.6 Control over scheduling algorithm................. 8-13

9. STORAGE STRUCTURE.......................................

9.1 Techniques in item and group level storage........... 9- 1
  9.1.1 Item level storage............................... 9- 4
  9.1.2 Item storage in groups and entries............... 9- 6

9.2 Techniques in entry and file level storage........... 9- 8
  9.2.1 Media and devices................................ 9- 9
  9.2.2 Storage structure organization.................... 9- 9
    9.2.2.1 Direct storage structure organization..... 9-10
    9.2.2.2 Relational storage structure organization. 9-11

  9.2.3 Indexes.. 9-13
  9.2.4 Relation of records to logic elements............ 9-14

9.3 Entry and file level structure....................... 9-15
9.4 Data base level storage structure.................... 9-26
9.5 User control of storage structure.................... 9-27

TABLE OF CONTENTS (continued)

Page

10. OPERATIONAL ENVIRONMENT.................................. 10- 1

10.1 Hardware environment..................................... 10- 1
    10.1.1 Processor, main memory, and special requirements.... 10- 2
    10.1.2 Data base storage media............................. 10- 3
    10.1.3 Terminal equipment.................................. 10- 3

10.2 Software environment..................................... 10- 3
    10.2.1 Operating environment............................... 10- 6
    10.2.2 Concurrency of operations........................... 10- 9
        10.2.2.1 Concurrency during file creation.............. 10- 9
        10.2.2.2 Concurrency with single copy.................. 10-10
        10.2.2.3 Concurrency with multiple copies.............. 10-12

    10.2.3 Modes of system use................................. 10-13
    10.2.4 Software facility interfaces........................ 10-14
        10.2.4.1 Operating system.............................. 10-14
        10.2.4.2 Communications subsystem...................... 10-16
        10.2.4.3 Other software................................ 10-20

    10.2.5 Procedure preparation, modification, and submission. 10-20

10.3 Modes of the operation by data base management functions... 10-22
    10.3.1 Batch mode.. 10-23
    10.3.2 Interactive mode.................................... 10-23
    10.3.3 Transaction mode.................................... 10-23

10.4 System transferability................................... 10-26

APPENDIX - INDEX.. A- 1

INTRODUCTION

I.1 Background

In May 1969, the CODASYL Systems Committee completed a report entitled "A Survey of Generalized Data Base Management Systems" [1]. This report contained a list of the features felt to be typical of such systems. For each of nine systems, a description was written in the format of the feature list. The report consisted of the feature list and nine separate chapters, one for each system described. The report was made available to attendees at the 10th Anniversary meeting of CODASYL (Conference on Data Systems Languages) held in Washington on 27 and 28 May 1969. The ACM assisted the Systems Committee by printing and selling copies of the report subsequent to the meeting.

I.2 Systems Committee goals

The next step in the Systems Committee's work was to evaluate how to pursue its goal of developing the specification of the common language and functions for a unified data base system. The word "common" is used to describe CODASYL developments in preference to "standard", because the development of standard systems is the responsibility of ANSI (American National Standards Institute). Hopefully, the common system would, after a number of implementations and a period of successful use, be submitted to ANSI as a candidate for standardization. This would then follow a path similar to that of COBOL which was developed by CODASYL between 1959 and 1961, submitted to ANSI (then ASA) for standardization in 1964 and accepted as an ANSI standard in 1966.

The committee, in planning its first step towards the goal, examined its own survey report to see how its usefulness as a basis for further work could be improved. For working purposes, the report was arranged by the committee so that each feature description was immediately followed by the nine narrative descriptions of that feature for all systems. The report was then divided into ten parts (corresponding to major sections of the feature list) for further detailed study by the members. It was agreed that a more comprehensive and complete feature list was required to reflect capabilities mentioned in some system descriptions which were not already included in the feature list. It was also agreed that the revised feature list should be validated by expanding the existing narrative on each feature to describe more completely the way a capability is handled in each system.

---

[1] References are at the end of this introduction.

The result of this work is a second technical report of which this introduction is the first chapter. However, it is the purpose of this introduction to serve not only as a prologue to the main report but also as a free-standing document in its own right for formal and separate technical publication. It reflects the opinions of the CODASYL Systems Committee as a group. Many of the issues covered in this introduction are those which the committee has discussed at length and on which a divided viewpoint must be recorded. This report attempts to indicate both sides of the picture where appropriate.

I.3 Decisions leading to preparation of current report

The current report on generalized data base management systems has been prepared and is being made available to the computer community only after considerable discussion and two somewhat divided votes on its content. Most members felt that the feature list used in the May 1969 report was not an adequate basis on which to design the common system which has been the committee's goal for many years. There was also widespread agreement that a more detailed feature list must be prepared based on a careful study of the resequenced form of the May 1969 report.

The issue which proved to be debatable was whether the committee should again publish information about specific commercial systems in its report. There was no question that such systems should be studied carefully to provide the necessary technical insight into the value and role of the various features; however, the merit of preparing and editing specific information about each system was open to discussion. On the one hand it was agreed that the Systems Committee should not give the impression that it was supplying a survey service for the computer community. On the other hand it was felt that a technical report containing only a feature list, with no tie-in to developed systems, would be too abstract and that inclusion of systems descriptions would give the report more technical content, and would focus more attention on the increasingly important role of generalized data base management systems. An important argument in favor of including the systems descriptions is the need to give a clearer focus on the differences among various systems.

The merit of including JOD COBOL [2] along with generalized data base management systems in the committee's report has been subject to some debate. COBOL does offer a data structure in the sense of the report, and it also has many of the features covered in the other sections. However it is not generally regarded as a generalized data base management system. The argument in favor of including it is that it provides a capability base against which to assess other systems. Many features provided in host language systems and self-contained systems do exist in COBOL.

It should be pointed out that the maintenance and development of COBOL is in the purview of the CODASYL Programming Language Committee.

I.4 Functions of current report

The latest report attempts to be more tutorial than its predecessor in the narrative description of each feature. Each feature description is then followed by either narrative information covering the ten systems or by a table for all systems. The systems covered in the report are CODASYL COBOL, as defined in the Journal of Development [2] , the Data Base Task Group's proposal [3] IBM's GIS, GE's IDS, IBM's IMS, Informatics MARK IV, IBM's FFS (also known as NIPS), Auerbach and Western Electric's SC-1, SDC's TDMS and RCA's UL/1.

The tabular approach is preferred in a minority of situations when the feature is sufficiently clearly identified across all systems so that the table does not require extensive footnotes to convey system capability. Where the narrative approach is used an attempt has been made to preserve a consistent level of detail across the systems included. This was lacking in the previous report because of the way in which it was prepared. However it should be pointed out that the format and mode of preparation of the previous report lead to a more cohesive total description of each of the systems included. If the present report were again resequenced to collect all narrative on an individual system together in one place, the resulting description of the system would not be as informative about that particular system as in the former report.

The function of the present report is to define more completely the features offered in present day systems. In this respect it is an extension of its predecessor. However it is intended that the feature list in the present report will provide insight and direction in developing design objectives for a common system, and that subsequently the report will be used to develop the specification of the common language and functions for a unified data base system.

I.5 Current state of the art

Generalized data base management systems are developed and marketed today under various generic names. Such appellations as data management system, generalized information retrieval system, information management system, and file management system are the main terms in use. The more elementary systems search a sequential file having simple record structures and provide only rudimentary report formatting facilities. More elaborate systems handle several files via indexes or links and function in an on-line mode. The number of ways in which these systems may vary is at least as numerous as the number of distinct features listed in the revised survey, and a user trying to select among the enormous variety of systems available faces a difficult evaluation problem.

The most significant point point observed in the course of the Systems Committee's study is the difference between host language capabilities and self-contained capabilities. The former are typically provided in systems such as IDS and IMS. The self-contained capabilities are found in systems such as GIS, MARK IV, NIPS/FFS, TDMS, and UL/1. Some members of the committee find it convenient to refer to two classes of systems - host language systems and self-contained systems. There is a divided opinion on the merit of this simple taxonomy, since some host language systems are now offering self-contained capabilities and some self-contained systems perform functions in a way more oriented to programmer use than to use by the non-programmer.

The difference between the two classes is one which is hard to delineate. The committee is generally agreed that the difference between system classes is one which ought to disappear and is in the process of doing so. There remains a difference between the two classes of capability which on the facilities level is easier to identify.

The host language capabilities may be identified simply as new tools for the application programmer. In this sense, they are embedded in a host language which is usually COBOL or PL/I. In addition, the facilities may be used by the assembly language programmer.

The self-contained capabilities are tools for the non-programmer as well as for the programmer. They are self-contained in the sense that they usually have no language connection with any procedural language. Procedural language in this report is used synonymously with procedure oriented language.

Before discussing each class of capability in more detail, it is important to note that many of the systems developed provide a data structure more powerful than that in COBOL. The committee concludes that such data structures are indeed required by the computer community today.

The way in which the data structure has been enhanced differs between the two classes, and also quite extensively within the systems providing principally host language capabilities.

Host language systems have either offered a more extensive hierarchical structure than that usually implemented for COBOL or else provide for the expression of network relationships among records. These subsume the hierarchical relationships more frequently provided.

Self-contained systems, at least those studied in depth by the committee, also provide for a more extensive hierarchical data structure than that of COBOL. However the extensions have usually been on the intra-record level and have included facilities for handling many levels of nesting and also variable length data items.

I.5.1 Host language capabilities

It is important to attempt to clarify the other differences between the two classes of capability. A system with host language capability is one which is built upon the facilities of a procedural language such as COBOL, PL/I or even basic assembly language; "built upon" in this context must not be confused with "built with."

To support the handling of the more complex structures, discussed in the previous section, facilities are provided to permit the user to initiate data transfers between the data base stored on low speed direct access memory, and high speed memory. Usually the implementation of the system embodies capability to avoid unnecessary physical transfers of data between memory levels, although the user of the system need not be aware of this.

The way chosen to interface the host language capabilities with the host language is usually through the CALL statement in the latter, although in some cases the host language has been enhanced to provide a neater interface. In general, however, the host language and its associated compiler have been treated as inviolate.

It is important to note that the user of a host language system is still to be considered an applications programmer in the sense that he writes a set of statements to be executed sequentially as in COBOL. He exercises almost the same degree of procedural control over the machine in his program as if he were programming in COBOL, except that the facilities of the data base system handle his data transfers once he has initiated them. He has control over the logical flow of his program and may mix conditional statements, action statements and loops virtually as he wishes. The enhanced data structure allows him a facility in handling certain data structures more completely than he would be able to in the host language alone. He is, of course, insulated from the physical storage structure, although required to assign a media type and define the file level structure for data base storage.

The data definition may be split into two levels. For example, the definition of items in records is performed in the host language as if no further system were involved. The definition of inter-record relationships is then performed in the facilities of the data base management system. In several host language systems these inter-record relationships are translated into an internal table or directory which the object program interprets at execution time. This interpretation gives a measure of data independence, a feature to be discussed later in this report.

I.5.2 Self-contained capabilities

The self-contained capabilities have developed from different origins and are aimed at handling a certain set of data base functions in such a way that conventional procedural programming is not required. A function in this set is always a high level one which might be programmed many times and which experience over the years has shown can be generalized - namely, programmed once with a high level language provided to express the various parameters of the function. The capability to mix conditions and actions as the programmer wishes is replaced by a pre-programmed or built-in processing algorithm so that the amount of writing required by the user is minimized. For this reason a system with such capabilities is sometimes called non-procedural, to indicate that the user does not exercise control over the sequence of detailed steps the system uses to process his requirements. Most significantly, he has no control over the sequence in which data are examined and moved from one level of memory to another or even from one area to another within high speed memory.

An important feature of systems offering self-contained capabilities is that the definitions of the data (on item, record and file levels) reside in some encoded form with the data. In fact, it may be stored in either a catalogue of several such data definitions or with the data file (namely, on the same volume).

The most commonly provided self-contained capabilities identified by the Systems Committee are the functions of interrogation and update. Interrogation is here defined to subsume the processes of data selection, sorting and report formatting. The interrogation function is one which has been very frequently generalized and has its origins in assembly language level report generators. The language of the interrogation function is sometimes called a generalized query language. Systems providing this kind of function are also referred to as generalized information retrieval systems.

The update function has been less frequently generalized, but any basically self-contained system which offers a data structure (within the file and record levels) more extensive than that of COBOL must also provide an update function, unless the files are to be updated by assembly language programs.

When either of the functions of interrogation and update are invoked, it is not necessary for the user to enter the data definition. This must have already been done, and the interrogation and update functions use the stored data definitions when they are executed.

Two other self-contained capabilities are identified as the functions of file creation and restructuring. An important function which must precede file creation is the process of translating the user expressed data definition into its stored form as mentioned above. File creation then includes expressing validation conditions which data entering the file must satisfy. File creation has the principal function of building the initial instance of the file, although in most cases this is essentially a use of the update function, discussed above, to update a null file.

Restructuring is the least frequently generalized function. It is basically a mapping of the data file performed by modifying the stored data definition and then mapping the records accordingly.

The disadvantage of systems offering only self-contained capabilities today is the size of the set of applications which they can handle. For those applications within the set, the self-contained systems offer considerably reduced set-up time and a vast reduction in the time required to prepare a new interrogation or update to the data base. In this respect, such systems can make major economies in the use of people's time and can also give a more rapid satisfaction of ad hoc information requirements. It is generally held that machine time spent using self-contained capabilities is longer than that which might be spent by a specially tailored program, depending on implementation.

I.5.3 Inter-system capabilities

Perhaps the biggest problem in the industry today is that the two approaches to data base management are not only incompatible with each other, but also are often data incompatible with the conventional procedural languages. This is primarily because both kinds of system provide data structures more complex than the procedural languages. Since the data structure enhancements vary considerably, it follows that systems within each class are also data incompatible with other systems in the same class. There are however some self-contained systems which are designed to operate only on the limited data structures of data files already being handled with widely used procedural languages.

Data compatibility is a problem which is steadily increasing in magnitude, and it is one which the two approaches to data base management are in effect making worse. Data compatibility across two software systems is a function of several levels of data storage. It is concerned with how data items are stored within a record, how logical records are stored in a physical block, how physical blocks are identified as belonging to the same file and how two or more files are stored on the same direct access volume. With further techniques such as primary and secondary indexes and inter-record pointers introduced, the likelihood of data incompatibility increases greatly. In fact, a COBOL source program can be transferred from one machine to another more easily than the data files on which the program operates.

The problem of data transferability is one which was debated at some length during the 10th Anniversary meeting of CODASYL held in May 1969. A data definition language was proposed as a solution to the problem, and a task group of the Systems Committee is currently studying storage structure definition languages.

I.5.4 Data independence and binding

Data independence is a capability frequently identified as required for data base management systems. The term "data independence" remains to achieve a widely accepted rigorous definition. It can be taken to imply some degree of insulation between a program and the data with which the program interacts. Depending on the extent of that insulation, programs can accommodate varying degrees of change in the definition and structure of that data, without it being necessary to modify the program or possibly recompile it.

Data independence can be achieved only by use of a stored data definition. If the program accesses (or interprets) this data definition at execution time in order to be able to locate items in records and possibly records in files, then this may be called execution time binding. The conventional approach to programming has evolved around binding the data to the program at compile time.

It should be noted that some host language systems offer a mixed approach to binding. The item level binding occurs at compile time but the record level binding occurs at execution time. This means that the tables or directories representing the inter-record level data definition (but not the intra-record level) must be accessible at execution time.

The self-contained systems rely heavily on the stored data definition. This does not imply an exclusive use of execution time binding, since object code may be generated, which implies that the data definitions are bound into that code at the time the statements in the interrogation or update are translated.

In summary, both host language systems and self-contained systems may in theory offer either approach (or even both approaches) to binding. By their nature the host language systems in use today have tended to offer the mixed approach described above, although there is no reason to adhere to this in future developments.

I.5.5 User interface

The next facet of generalized data base management systems which should be mentioned is the user interface or, more explicitly, the language which the user has to learn in order to use these systems. In the host language systems, the applications programmer must learn the enhancements which may be new elements of the host language, or more likely a set of parameters to the CALL statements.

In the self-contained systems, a complete new language is offered. If this is used by the non-programmer for whom it is intended, then the numerous inconsistencies with procedural languages are not embarassing. To the individual with a smattering of COBOL, any use of the self-contained function will normally involve less writing than he would otherwise have to do in COBOL. This situation is due to the fact that the file processing algorithm is built into the function he is invoking and does not have to be spelled out.

With many of the more powerful self-contained systems available today it is clear that little attempt has been made to stay close to the statement forms provided in COBOL, even where it is feasible to do so.

I.5.6 Levels of user

Four levels of user may be identified - data administrator, applications programmer, non-programmer and parametric user.

Both classes of systems identify the role of an individual called the data administrator. In a user environment in which an important data base is kept on-line for access by several individuals, there must be a single individual who carries responsibility for many facets of its use. Certainly he should be responsible for its initial creation and for instituting any structural modifications which may be required. In identifying specific features of the generalized data base management systems as restricted to privileged use by one individual, the Systems Committee is expressing a collective opinion on what such features should be. The use of the system features provided for the data administrator may call for considerably different levels of expertise in the different systems.

The next level of user is the applications programmer for whom the host language capabilities are specifically provided. He is well identified in terms of current practices. However, when programming to operate on data which is stored on-line under the management of a generalized data base management system, he may have to accept constraints and disciplines which have been noticeably absent in previous methods for handling business applications.

A third level of user is the non-programmer for whom the self-contained capabilities have been designed. Finally, is the other non-programmer level which is of importance only in an on-line environment. He may be called the parametric user, since his interface with the data base is one of invoking pre-defined transactions and possibly providing values to any parameters they may have.

I.6 Technical problems facing designers

There are many technical issues facing future designers of generalized data base management systems. There exists a broad spectrum of systems which could, in theory, provide them with a wealth of input. Unfortunately, with the tremendous investment in software development taking place today, and due to the unprecedented rate at which computer science is evolving compared with other more mature disciplines, it is very difficult to recognize the mistakes of others - let alone learn from them.

One of the reasons that the Systems Committee has spent a period of three years largely devoted to the study of existing generalized data base management systems, is to try to chart a path for the future of such systems. Although it is not the purpose of this document to spell out detailed guidelines for future systems, a number of pointers can be given.

I.6.1 Existing storage structures

First, there is an enormous investment today in stored data. While it is virtually impossible to develop a system which could process all such data, it should be one of the functions of a new system to act on the commonest storage structures in use. This could possibly be by means of run-time interpretive access to the data. This approach is preferable if the data is to be accessed more frequently by some other system with which it may have been bound at compile time for more efficient access. If the future accesses are considered to be more important than those in the past, then a one-time conversion of the data may be more efficient. The problem of facilities for accessing commonly used storage structures is a significant one for future designers.

I.6.2 More complex data structures

It is apparent that more complex data structures must be provided. Certain applications require intra-record and inter-record structures more complex than those currently provided in COBOL. The Data Base Task Group (DBTG) of the CODASYL Programming Language Committee released a report [2] in October 1969 proposing a data structure for storing records in a file which permits the user to represent network or graph structures. This report has been considerably amplified since October 1969 and a revised report should be available in June 1971.

The DBTG proposal consists of two parts. The first is called the DDL, Data Definition Language, in which such network structures may be defined. The second is the DML, Data Manipulation Language, which is used to operate on the data records stored in such structures.

The DBTG considers its separation of the DDL and DML to be the cornerstone of its approach, in that the separation allows data bases described by the DDL to be independent of the language or languages used for processing the data. The DML proposed in the October 1969 report augments COBOL and thus the Systems Committee considers the DBTG proposal as a host language system and has included it as one of the systems in its new report. The DBTG anticipates the DML will be used to augment other languages such as FORTRAN and PL/1 and that, in the future, self-contained languages will also be developed to operate on data bases described by the proposed DDL.

I.6.3 Self-contained operations on network structures

It is significant that the self-contained systems typically operate only on purely hierarchical structures and not on the more complex network structures. The reason for this is that the former are more widely understood and used. It is debatable what percentage of applications in fact call for a network capability. However, these are certainly numerous enough and important enough to justify the introduction of network structures and more and better understanding must lead to a more widespread use. Presumably, when such techniques have been as widely used as simple sequential files are today, it should be possible to define higher level functions, such as those in the self-contained systems,to operate on these structures.

I.6.4. Non-programmer interaction

Another major issue facing the designers of generalized data base management systems stems from the increased interest in allowing non-programmer users to interact directly with a large shared data base. This interaction is generally on-line from a terminal (for which case the use of the ill-defined term "time sharing" should be avoided), but may be by means of batch input and output. There are two classes to be considered here.

The first is the parametric user referred to earlier, who invokes a pre-stored procedure and possibly provides certain data values to that procedure. One such occurrence - invocation and input - is called a transaction. The term transaction program is used for the pre-stored procedure, and transaction input for the data itself. The function accomplished by a transaction program may be either interrogation or updating; the usage of the term in banking and accounting is generally limited to the latter. The parametric user has probably not had any hand in preparing the transaction program and does not know the language in which it has been written. His understanding is limited to the types of transactions which he has been taught how to invoke, and the meaning of any output, including error messages, they may generate.

Some host language data base systems provide on-line capability to handle this kind of user. When the volume of transactions is high, special transaction management schemes are required. Such schemes are quite widely used in environments where everything is tailored to the specific application, such as airline reservation and credit checking. The emphasis is on a very high volume of transactions with possible priority schemes for queuing and routing of the transactions. Such capability, while sometimes subsumed under the name of the data base system, is in fact an area of expertise in its own right and has not been examined closely by the Systems Committee.

The other class of non-programmer user is the one who formulates his own query (or update) and hence needs to understand the language in which he must do this. He may be specifying his query in a truly interactive mode and carrying on a dialogue with the syntax processing part of the system. While making his query syntactically correct for the data files and their indexes, he may at the same time narrow his question to the one he wishes to ask by a browsing process.

Since the self-contained systems have a clear orientation towards facilitating the asking of ad hoc questions, this class of systems is having more impact on the individual who formulates his own query.

I.6.5 Unification of both approaches

The development of a unified system to support the parametric user and the non-programmer is a major problem facing the designers of generalized data base management systems. It is important that facilities for both users can operate on the same storage structures. The present situation is one in which a predefinable transaction program is normally prepared in a host language system but an ad hoc question is posed using the facilities of a self-contained system. In unifying these two approaches, it appears desirable to many members of the committee that it should also be possible to prepare a transaction program using self-contained capabilities where these are adequate and host language capabilities where the transaction program needs facilities not available in the self-contained capabilities.

Not only is it important to allow the same data base to be created and updated by several levels of languages, including the procedural language and the non-procedural user oriented languages, but there must be a separation of the data definition process from the languages which operate on the data thereby defined. In other words, the data definition must be common to a number of languages.

I.7 Use of COBOL as a basis for further development

The future role of COBOL in the data processing community is one which has been of some concern to all CODASYL committees. Nobody can deny that COBOL is currently the most widely used programming language in the world today, a status which it seems to have achieved in a period of less than a decade. However, when compared with FORTRAN this fact reflects the predominant use of computers for business application over scientific. When compared with PL/I, this fact reflects COBOL's four or five year start in life.

On the other hand, COBOL has been frequently criticized on a number of counts. Many say it is overloaded with features. Others critize its verbosity, while still others allege greater elegance of both PL/I and ALGOL.

It must be observed that programming languages used by man to communicate with machines are becoming as difficult to replace by legislation as the natural languages used by him to communicate with his fellows. The English language has been widely criticized on the basis of its illogical spelling, and its ambiguity, but it is the most widely spoken language in the western world. From this perspective it seems unlikely that COBOL will be successfully replaced overnight, although it has to change to accommodate new requirements. The CODASYL Programming Language Committee has been vigorously pursuing its enhancement and examines hundreds of proposals every year.

I.7.1 Data structure capabilities

It is a debatable issue how far one can go in using COBOL as a base for developing a common generalized data base management system. The data structures of COBOL, while appropriate for many types of batch processing, are felt by many to be too specialized for data base applications. As mentioned earlier, the Data Base Task Group has proposed a more general class of data structure in which network or graph like structures may be defined. Such structures permit a more direct representation of the complex relationships among the entities of a business enterprise, and permit many different logical orderings to be defined instead of the single ordering implied in a hierarchical structure.

The development of many generalized data base management systems has shown that enhancements to the data structure on the record and item levels have been found necessary in addition to those proposed by the DBTG. For example, many systems handle complex intra-record structures to permit processing of variable length data items which would typically be used for lengthy string data. Such enhancements to the data structure also require string processing facilities to permit character level operations on such data.

It must be pointed out that the capabilities offered by the DBTG's proposed inter-record structures and many other systems intra-record structures are not mutually exclusive in terms of the complexity of data structure which can be represented. Inter-record level facilities lead to smaller simpler records and more logical accesses; intra-record level facilities lead to larger more complex records and fewer accesses to larger units of data.

I.7.2 Host language capabilities

The host language facilities provided in a generalized data base system could be provided by COBOL, suitably augumented to reflect the system's data structure class as indicated in the preceding section, and also with the Procedure Division enhanced to provide other features needed for writing and executing programs in a shared data environment. Such extensions to the Procedure Division have been proposed by the DBTG in the form of the Data Manipulation Language (DML).

As an alternative to such substantive additions to COBOL, it is possible that COBOL and other existing procedural languages could be used for data base application without enhancement by permitting the user to define mappings between data base structures and application program structures. This approach would allow the applications programmer to select a data structure class appropriate to his program, rather than one of a prescribed set of data structure classes. This could simplify the task of obtaining agreement among the developers of different programming languages for a consistent set of enhancements. The DBTG's subschema concept provides this approach to mapping, and a full application of the technique remains to be fully explored.

I.7.3 Self-contained capabilities

There has been no cooperative industry-wide effort to date to address self-contained capabilities. COBOL development and the DBTG proposal have both addressed the provision of capabilities to the applications programmers. The importance was stressed earlier of allowing a data base to be interfaced by both host languages and self-contained languages. The DBTG approach provides a DDL which allows describing data independently of the languages used to process that data. By permitting the referencing of external data definitions, both host languages and self-contained languages, existing and newly developed, could interact with the same data base. The specification of self-contained languages suitable for this purpose remains to be investigated.

Using COBOL as a basis, the provision of the typical self-contained capabilities, such as interrogation and update, might be achieved by identifying each as new divisions which could operate on the data structure previously defined in the Data Division by a data administrator. A stored representation of the data definition may or may not be stored with the data. Such capabilities could be on the level of those found in the self-contained systems, with all or most file processing built into the function. A similar approach was suggested by the DBTG in its proposal which also indicates that the Data Division should be enhanced to permit the naming of an external data definition expressed in the DBTG's DDL. It could then be possible for other languages outside COBOL and its non-procedural facilities, to operate on the data base, as described previously in the discussion of the DBTG work.

I.8 Features of generalized data base management systems

This report covers the features of generalized data base management systems in ten chapters. The ordering of these chapters was motivated by a goal to present the feature analysis to an audience with a wide range of backgrounds. Some readers may have no experience at all with data base management systems while others may have either implementation or user experience with one of the two principal classes discussed in this introduction.

Following a general summary chapter, the next two deal with the capabilities of the systems to represent data and then an analysis of the language forms used to define the representable data structures.

An understanding of the data structure concepts introduced in Chapter 2 is important to an understanding of the following chapters which deal with functions and facilities for acting on the data. Much of the terminology used in the report is also developed in this chapter.

Chapters 4 and 5 analyze the self-contained functions of interrogation and update and chapter 6 discusses the ways in which a file is initially created. Chapter 4 and 5 follow the same general outline as in the previous report although considerably extended and more thorough.

There was considerable debate by the committee on how to present a feature analysis which did justice to both self-contained and host language systems. The idea of preparing two separate reports, one for each class, was discussed but it was realized that there would be major overlap and such an approach this might tend to promulgate an undesirable dichotomy. Following the decision for a single report, two chapters were added. Chapter 7 is a major new chapter containing many additional features not included in the previous report. It deals with facilities for the programming user which are provided through a procedural programming language. Chapter 8 identifies the functions ascribable to the data administrator. The capabilities of the systems known to the various levels of user are therefore user-oriented and covered in Chapters 2 to 8. This is in contrast to the last two chapters which deals with systems oriented features which are often transparent to the user. Chapter 9 analyzes the various approaches to storage structure, which is not necessarily known to the user and chapter 10 outlines the operating environment under which each implementation runs. This last chapter is not relevant to the Data Base Task Group proposal, nor for JOD COBOL for which no specific implementation is considered.

The following is an outline of each of the ten chapters:

I.8.1 General summary

The opening chapter identifies a set of major features which convey an idea of the class of a system. Its inclusion is chiefly to provide identification, background and reference material for each system included in this report. This chapter also gives an overview for each system and is the only one which deals with each system in its entirety.

I.8.2 Data structure

Data structure is the view of the data as seen by the user of the system and excluding any details of storage techniques used which are covered in a separate chapter. An understanding of the data structure of either kind of data base system is essential to a good understanding of its capabilities. As indicated, most systems have provided a data structure capability different from that of COBOL although the differences are on different levels.

The Systems Committee found it important, even in its May 1969 report, to differentiate between the logical data structure which the user of the system must understand, and the physical storage structure which sometimes takes the role of the implementor's trade secret.

The present report's features listed under data structure handle hierarchical tree structures and the kind of network structures in the Data Base Task Group's proposal.

Data structure levels are identified as item, group, group relation, entry (record), file and data base. The definition of a data structure is referred to throughout the report as a schema. It is also possible in some systems to have several sub-schemas which are subsets of the schema.

I.8.3 Data definition

This chapter is tied in closely with the previous one, the difference being that this discusses the language and/or tabular formats used to define a schema representable within the system's capability to handle data structures. The definition of each level of data in the data structure is discussed in the same sequence as it is introduced in the previous chapter. This chapter also discusses the entering of the data definition and the important concept of binding.

I.8.4 Interrogation

Interrogating a data base is a process of selecting and extracting some part of the whole data base for display, usually in a hard copy printed form. One section of the interrogation function defines how the part is selected. The second part covers how operations such as computation, sorting and formatting may be performed on the selected part. The concept of interrogation is an intrinsic self-contained capability. The implication is that the user is able to formulate a query in the language of the system without detailing the sequence of steps used to access the data base and extract the information.

Availability of the interrogation function implies that a built-in processing algorithm for the function is provided by the system. In the simplest case, the processing algorithm is that of sequentially searching a stored file, copying out records which satisfy some conditional expression, and building up a report based on the data contained in these records. There are many degrees of sophistication even within the framework of the basic sequential search algorithm. Other processing algorithms cause the file to be accessed to obtain the required information, using various techniques which avoid a sequential search.

I.8.5 Update

Updating a data base is a process of changing the value content of some part of the data base. It excludes restructuring of the data which would cause a modification to the stored data definition. Update is a process somewhat analogous to interrogation in that some part of the data base must first be selected. In most self-contained systems, the selection facilities are modelled on those used in the interrogation function. However, once the part is selected, it is changed in some defined way rather than displayed in a report.

Update is intrinsically a self-contained capability. It also implies a built-in processing algorithm, but the possible ways of implementing it are even more varied than for interrogation. In some systems, both update and interrogation can be performed during the same sequential pass of a file in the data base.

I.8.6 Creation

An important preliminary to the creation function is that of data definition. It is necessary to provide a set of records to form the initial instance of a file. Other functions are data validation, security specification and control over media type. Data base creation is considered to be one of the important functions for the data administrator. Creation may imply a built-in processing algorithm as for interrogation and update, or it may have to be programmed in a conventional sense. In many cases it is a use of the updating function applied to a null file.

There is no clear division here between self-contained systems and host language systems. Some self-contained systems do require a programmed approach to file creation. This implies that providing the initial instance of the file is a function which has to be programmed using facilities other than those provided by the system.

I.8.7 Programmer facilities

Programmer functions are defined as host language capabilities. They are functions upon which a programming user may call when writing a program in a host language. The most important programmer function statements are those which permit him to initiate data transfers between the stored data base and high speed memory. Other statements may be provided to allow him to issue file control statements such as open, close and hold.

Any function considered to be in the domain of the data administrator, even though its use may be on the level of the programmer, is not considered in this chapter. Also, facilities in self-contained systems for linking to procedural language functions (such as own code hooks) are not considered in this chapter.

I.8.8 Data administrator functions

The data administrator is an individual responsible for a data base. His role is identified to some extent in both host language and self-contained systems. The important functions of data definition and file creation are each covered in a separate chapter, but there are other functions which are ascribed to the data administrator. Such functions include monitoring system operation, preservation of system integrity and security, and providing for restructuring the data base to accommodate new record types or new items. Some of the data administrator's functions may have to be performed with a programmer level language in some systems. In this case the designation of a function as an administrator function is subjective.

I.8.9 Storage structure

Each level of the data structure has a stored representation which is referred to as the storage structure. The file level storage structure defines how entries are stored in physical blocks to form the stored representation of the file. This level is often dictated by the input/output control system, which in third generation operating systems has been given the name of data management system. File level storage structures include such techniques as indexed sequential and other ways of storing a file and data about it to facilitate access to its contents.

The entry level storage structure varies more widely among systems and it defines how groups or items are represented in storage to form the stored representation of an entry. Sometimes all entry data is stored contiguously in low speed memory, but in some systems groups are mapped into segments where the segments in an entry may be stored in different locations in low speed memory. Finally item level storage structure usually reflects the storage modes of the machine although systems exercise different levels of control in their data structure over the mapping of items into storage structure formats.

I.8.10 Operational environment

All systems which have been implemented have one or more operational environments in which they function. This environment consists of a hardware configuration and a software environment usually provided by the operating system. This chapter is directly relevant to the capabilities of the systems themselves only to the extent that it explores the interface with other software components and the concurrency with which the various functions can be executed.

I.9 Report conventions

I.9.1 Terminology

In carrying out an analysis of different systems the problem of disparate terminology used remains a major problem. The terminology base developed in the Systems Committee's previous report [1] appears to be having some influence in the industry, but development of most of the systems analyzed for this current report was fairly advanced or even completed by the time the previous report was released.

Where appropriate, the current report uses essentially the same terminology as the previous one, with a few minor modifications. However this report contains a far more detailed and broader discussion of the features and capabilities of data base management systems, and terms are frequently needed to convey concepts not identified at all in the previous report. An attempt has been made to use widely used terms where it is felt that these are indeed established and accepted. The terminology of the CODASYL Journal of Development [2] of the Data Base Task Group's proposal [3], have been adopted where applicable, although there is some deviation in the use of certain terms, for instance "schema."

Most of the terminology is introduced in Chapter 2 and the appendix contains a list of terms with a reference to where in the main body of the report an explanation can be found.

I.9.2 Language syntax specifications

Where examples of language statement types are given, the following rules are followed :

1. The elements which make up a clause or statement consist of upper case words, lower case words, level numbers, special characters.

   Example: ABCD, abcde, 01, =, ⇛

2. Upper case words appear exactly as shown. No distinction is made between required words and noise words.

3. An asterisk indicates a default option.

4. Lower case words are metalinguistic elements which must be replaced in a program by appropriate names or values.

5. The specific level-numbers used in describing the Record Section entry formats of the DBTG and the COBOL Language are required when such entries are used in a program.

6. The meaning of enclosing a portion of a general format in special symbols as follows is:

   $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$ at least no occurrences
   at most one occurrence

   $\begin{Bmatrix} a \\ b \\ c \end{Bmatrix}$ at least one occurrence
   at most one occurrence

   $\begin{Vmatrix} a \\ b \\ c \end{Vmatrix}$ at least one occurrence
   at most one occurrence of each

7. An ellipsis (...) indicates repetition is allowed in the source program. The portion of the format which may be repeated is determined by the [or { which logically matches the] or } to the immediate left of the ...

8. The symbol := means "is defined as", for example

   on-clause:= ON ERROR PERFORM

I.9.3 Abbreviations

In tables and figures "n.a." means "not applicable."

I.10 Systems Committee role in future development

As was pointed out earlier in this paper, the CODASYL Systems Committee has spent three years in an intensive study of generalized data base management systems. The fact that this study has taken so long is an indication of the ever increasing complexity of the topic. The fact that numerous organizations have seen fit to sponsor members' participation over such a period of time can be taken as an indication that an in-house understanding of these systems is of utmost importance to the affiliations concerned.

The reason that the Systems Committee has chosen to publish two comprehensive surveys is to make insight available to the data processing community. The members who have participated in this effort feel that they have developed considerable insight into the possible capabilities of generalized data base management systems. One aim in publishing the two reports has been to attempt to disseminate some of this insight and to try to channel thinking and understanding of such systems into a generally acceptable form.

The charter of the Systems Committee is stated in the CODASYL constitution reads: -

> "The Systems Committee strives to build up an expertise in and to develop advanced languages and techniques for data processing, with the aim of automating as much as possible of the process currently thought of as systems analysis, design and implementation."

At this point in time, the expertise in this all important area can be said to be built up. The next step is the specification of a common system and the Systems Committee hopes that the insight it has documented can contribute towards the development of such a system.

References

[1] A survey of generalized data base management systems. CODASYL Systems Committee. May 1969. Available from ACM, New York City and IFIP Administrative Data Processing Group, Amsterdam at $7.00.

[2] CODASYL COBOL Journal of Development 1969. Number 110-GP-1a. April 1970. Available from Canadian Government Specifications Board, Ottawa, Canada.

[3] CODASYL Data Base Task Group Report. Available from ACM New York City, IFIP Administrative Data Processing Group Amsterdam, and British Computer Society, London.

1. GENERAL SUMMARY

This chapter provides a technical introduction to the feature analysis of generalized data base management systems. Ten systems have been used as a basis for the analysis, including the COBOL programming language and the Data Base Task Group's (DBTG) April 1971 proposal. As background to the feature analysis, the systems are identified and a brief summary of their salient features is presented. This summary includes a general classification of these systems, the data structure class they deal with, the generalized processes provided and how they are achieved, the forms and types of system language(s), the storage structure class and the hardware and software environment in which these systems are implemented. Since this chapter is the only one which deals with the systems as an entity (the remaining chapters only deal with them with respect to their features), a system summary is presented including a brief statement of the system development philosophy and a narrative description of each system. Finally the last section provides a selected bibliography on the systems.

1.1 Identification

A significant distinction is noted between system capabilities which are provided through system language(s) tailored to particular functions and capabilities which are provided by augmenting a general purpose language such as COBOL or PL/1. Capabilities in the former category are labeled self-contained capabilities, and those in the latter category are called host language capabilities.

1.1.1 Self-contained

Systems in the self-contained category date back to the early nineteen-sixties. These systems typically are at least "second generation" systems, each having at least one precursor system while one system has been through at least three iterations. Figure 1-1 identifies the self-contained systems used for the feature analysis of this report by providing their name, acronym, originator, and initial release date.

IDENTIFICATION	ACRONYM	ORIGINATOR	INITIAL RELEASE
Generalized Information System	GIS	IBM	1969 September
MARK IV	MARK IV	Informatics Inc.	1968 (full system)
NMCS Information Processing System	NIPS/FFS	DCA, IBM, National Military Command System	1968 July
Time-shared Data management System	TDMS	System Development Corporation	1969
User Language/1	UL/1	RCA	1969

Figure 1-1
Self-contained systems

1.1.2 Host language

Most systems which provide a host language capability have been implemented or specified during the last few years. Systems in this category augment the basic procedural language capabilities with data manipulation statements, and the data structure class of their host language (e.g., PL/I or COBOL) is extended to handle the more complex plex data structures. As a reference point for this category, two additions have been made -- JOD COBOL and the DBTG proposal. While the DBTG proposal specifies the necessary components for a data base management system, the COBOL language usually is not considered a data base management system. Figure 1-2 identifies those systems which provide a host language capability whose features are analyzed in this report.

IDENTIFICATION	ACRONYM (as used in report)	ORIGINATOR(S)	INITIAL RELEASE DATE
Journal of Development Common Business Oriented Language	COBOL	CODASYL	1970 April (specification)
Data Base Task Group Proposal	DBTG	CODASYL Programming Language Committee	1971 April (specification)
Integrated Data Store	IDS	Honeywell Information Systems	1963
Information Management System	IMS	International Business Machines North American Rockwell	1969 September
System Control-1	SC-1	Western Electric Auerbach [1]	1970 July

[1] Developed by Western Electric with Auerbach providing the technical assistance based upon the DM-1 concept.

Figure 1-2
Host language systems

This dichotomy between systems that provide host language capabilities and those that provide self-contained capabilities is by no means exact, nor are the two categories mutually exclusive. It is possible for some systems which provide a host language capability to also provide self-contained capabilities. It is also possible for a self-contained system to provide a host language capability.

1.2 Data structure class

Data structure is the users' conception of the data, independent of the way in which the data are stored (storage structure) by the system. It provides a picture of the data base so that the structural components and their attributes can be described. The data structure class indicates the data structuring capability of a system by describing the class of data structures made available to the user. The data structure class is defined by a fixed set of generic structure types: item, group, group relation, entry, file, and data-base. The entry level is chosen as the basis for summarizing the data structure class because it is usually used to represent the major entities of an application (e.g., the employees of a firm). An entry is a set of groups and group relations in which one and only one group, the entry defining group, is not contained in or subordinate to any other group. There are three types of entry identification: a group entry which consists of a single compound group in which the subordinate groups are nested; the tree entry which is a set of simple groups (non-nested) possessing a hierarchic group relation; and the generalization of the tree entry the plex entry -- a set of group relations in which every group except the entry-defining one can be subordinate to another group and participate in more general relations. Figure 1-3 summarizes the data structure class by indicating the types of entries and files of the systems analyzed in this report.

SYSTEM NAME	DATA STRUCTURE CLASS	
	ENTRY TYPE	FILE TYPE
GIS	tree	unlinked
MARK IV	tree	unlinked
NIPS/FFS	group	unlinked
TDMS	group	unlinked
UL/1	special[1]	unlinked
JOD COBOL	group	unlinked
DBTG	group	linked
IDS	group	linked
IMS[2]	tree plex	linked
SC-1	group	unlinked

[1] UL/1 allows up to 15 sets of group relation schemas each structured as a tree

[2] IMS has two data structure classes: one for the data administrator (linked) and one for the application programmer (unlinked), the latter being a subset of the former.

Figure 1-3
System data structure class

1.3 Generalized processes provided

A generalized data base management system is potentially capable of providing generalized processing facilities for either the programming user or the non-programming user, or both.

The non-programming user is not necessarily a non-programmer. This term is used to indicate that the user is not required to write a program in a conventional programming language in order to use the data base. In this sense, users are being described according to what they have to do as opposed to what they have to be (their skills, etc.). For the non-programming user, the system may provide facilities for performing such functions as data definition, file creation, interrogation, extraction, update, and data structuring. The user invokes the function and provides the input it requires which could be anything from a small set of parameters to a "program" written in some special purpose language. A self-contained system usually provides a special language for the user to specify high level operations oriented to the function to be performed.

The facilities provided by a system for the programming user are called data manipulation facilities, because they are directed toward the explicit manipulation of a data base by a programmer. The programmer data manipulation facilities often consist of such operations as opening and closing files, fetching or storing data on direct access storage devices and searching for, holding or modifying particular sets of data. These facilities are requested through statements embedded in a procedural program written in a conventional programming language, called the host language.

It is possible for a system to provide facilities for both categories of users. For the non-programming user, program modules are designed to perform common functions with a special interface -- usually some form of command interface to call upon the data manipulation facilities. If a system with the self-contained functions also provides programming facilities, perhaps the same as are used by the function modules, programming users are then able to create their own program modules to do special tasks, to perform normal functions (update, interrogate) in special ways, and to tailor-make their own application programs.

Although there is no universal agreement on all of the generalized functions provided by the host language and self-contained systems, the functions deemed part of a generalized data base management system are data definition, file creation, file update, interrogation, and a programming facility. All of the systems possessing self-contained capabilities analyzed in this report provided all of the generalized functions except a programming facility. On the other hand, all of the systems manifesting host language capabilities provided (by definition) a programming facility, and a data definition function but not usually the self-contained functions of creation, update and interrogation. One system, SC-1, provides to a large degree all data base management functions.

1.4 Language type

Systems, particularly those with self-contained capabilities, often have multiple languages or at least several sub-languages. The purpose of the language is to provide a facile user interface to the system; that is, the intention is to make it easy for the user to use the capabilities of the system. Typically these languages exist in many forms and levels depending on the systems. Four types are identified as narrative, keyword, separator, and fixed position. Usually the system languages are not pure but rather are a mixture of the different language types.

The most general kind of language form is free form narrative, sometimes called English-like. Generally there are severe syntactic restrictions, but the result is something resembling sentences as in the example from the DBTG proposal:

```
SCHEMA NAME IS EMPFILE

RECORD NAME IS EMPREC

LOCATION MODE IS CALC USING EMPNO

SET NAME IS SKILLS

PRIVACY LOCK FOR REMOVE IS AUTHENTICATE
```

In many cases additional "noise words" may be included for clarity but are ignored by the system. In the previous example the noise words THE and FUNCTION can be added to the last phrase yielding:

```
THE PRIVACY LOCK FOR THE REMOVE FUNCTION IS AUTHENTICATE
```

The keyword form, the second language type, consists essentially of a sequence of attribute-value pairs, as shown in the following example from a GIS data definition:

```
FLD:NAME = EMPNAME, UNITS = PACD, LENGTH = 20,

     HEADER = EMPLOYEE NAME

SEGM:NAME = EDUCATION, LEVEL = 2
```

Languages for systems which operate in an interactive mode are usually of this type:

The system outputs - FLD:NAME =
and the user responds - EMPNAME

This has the advantage (shared by the questionnaire-type systems) of guiding the user through the preparation process.

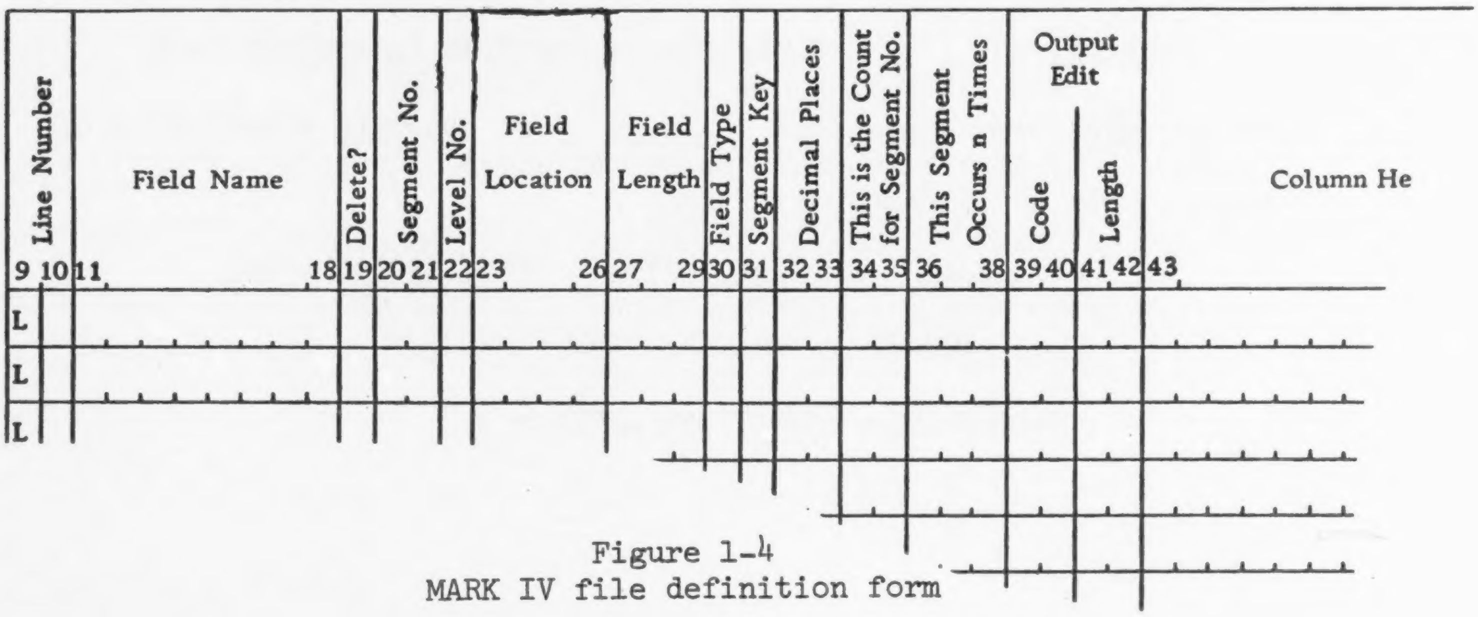
If the set of attributes and their sequence is fixed, the separator form is used. Only the attribute values, separated by some special character are required as input. The separator form usually requires a "context specifier", a keyword followed by a sequence of attribute-values. Thus the separator form appears to consist of clauses which begin with a keyword, sometimes a verb, followed by parameters or modifiers in separator form. It is evident that separator form becomes keyword form when only one value follows the keyword.

The following item definition in the separator form is taken from GIS and is the same example as used in the keyword form, except the units data (PACD) is omitted (indicated by the two adjacent commas). This indicates the multiple language forms of the system's data definition capability.

FLD, EMPNAME,, 20, NAME

In this case the separator is the "comma."

Finally there are those systems in which each element of the definition appears in a fixed position (e.g., punched card column) on an input medium. Often a preprinted form (or questionnaire) is provided to simplify the user's task, as shown in the example of a MARK IV partial file definition form in Figure 1-4.



Line Number 9 10	Field Name 11 18	Delete? 19	Segment No. 20 21	Level No. 22	Field Location 23 26	Field Length 27 29	Field Type 30	Segment Key 31	Decimal Places 32 33	This is the Count for Segment No. 34 35	This Segment Occurs n Times 36 38	Output Edit Code 39 40	Output Edit Length 41 42	Column He 43
L														
L														
L														

Figure 1-4
MARK IV file definition form

Figure 1-5 summarizes the various language forms used as the interface to the data definition, interrogation and update functions.

SYSTEM	FUNCTION			PROGRAMMING FACILITIES
	DATA DEFINITION	INTERROGATION	UPDATE	
GIS	all forms are provided	narrative	narrative	n.a.[1]
MARK IV	fixed position	fixed position	fixed position	n.a.
NIPS/FFS	separator	separator	narrative(NFL)[2] keyword (OM) fixed position (POOL)	n.a.
TDMS	narrative	narrative	narrative	n.a.
UL/1	separator; some narrative elements	narrative	narrative	n.a.
COBOL	narrative	n.a.	n.a.	narrative
DBTG	narrative	n.a.	n.a.	narrative
IDS	narrative	n.a.	n.a.	narrative
IMS	separator or keyword	n.a.	n.a.	separator
SC-1	separator	narrative	narrative	narrative

[1] not applicable

[2] NIPS/FFS update languages

Figure 1-5
System language types

1.5 Storage structure class

Storage structure is the implementation of a particular data structure on the physical storage media. Whereas the data structure is the user's view of the data, the storage structure is the system's view of the data. Storage structure can be described informally as the process of relating the instances of the data structure components to the physical components of the store (file, records, tracks, blocks, words, bytes).

Table 1-6 presents the file level storage structure and indicates if the system provides user-controlled secondary indexing.

SYSTEM NAME	STORAGE STRUCTURE CLASS	INDEXING
GIS	sequential (fixed, variable) indexed sequential (fixed)	
MARK IV	sequential (variable) indexed sequential (fixed, variable)	
NIPS/FFS	sequential (variable) indexed sequential (variable)	
TDMS	cross indexed tree structure completely inverted	complete[1]
UL/1	sequential, variable length	
COBOL	sequential or random	
DBTG	strategy pointer array	yes
IDS	forward pointers, and optionally, backward pointers and pointers to parent no group cycles	
IMS	IMS entry sequential indexed sequential direct indexed direct } chaining	
SC-1	sequential (fixed blocks)	yes

[1] The storage structure class is completely indexed; hence there is no need for a secondary indexing capability.

Figure 1-6
System storage structure class

1.6 System environment

No description of a data base management system is complete without a summarization of the operational system environment -- both hardware and software -- in which the system functions. A key implementation consideration is the interface between the operating system and the data base management. An important user consideration is the mode or modes of operation available to him. Figure 1-7 summarizes the system environment of the data base management system by indicating the latest version of the software, the hardware including the smallest configuration in which it will operate, the operating system (version and type) and the modes of use of the system.

SYSTEM	ENVIRONMENT				
	HARDWARE	OPERATING SYSTEM	SMALLEST MACHINE	MODES OF USE	
				BATCH	ONLINE
GIS	IBM System 360	O/S 360 MFT-II MVT	Model 40G Model 50I	yes yes	yes yes
MARK IV	IBM System 360 RCA SPECTRA 70	O/S 360 DOS 360 PCP TDOS	Model 25J Model 45	yes yes	no no
NIPS/FFS	IBM System 360	O/S 360 MFT-II MVT PCP	Model 40H Model 50I	yes yes	no yes
TDMS	IBM System 360	ADEPT 50	Model 50H	no	yes
UL/1	RCA SPECTRA 70	TDOS	Model 45	yes	no
COBOL	n.a.			yes	yes
DBTG	n.a.			n.a.	n.a.
IDS	H 600 and H 6000 series	GECOS-III	H 615	yes	yes
IMS	IBM System 360 and System 370	O/S 360 MFT and MVT	Model 50H Model 145H	yes no	no yes
SC-1	IBM System 360	O/S 360 MFT MVT	Model 50H	yes	no

Figure 1-7
System environment

1.7 System summaries

The design philosophy and a narrative system description of the eight systems usually identified as data base management systems, JOD, COBOL and the DBTG proposal are presented in this section. The purpose is to present an overall view of the systems whose features are analyzed in this report, but not necessarily to provide total insight into these systems. The design philosophy is followed by an overall narrative description of the systems analyzed.

1.7.1 Underlying philosophy

In the design of data base management facilities, particularly the self-contained functions and the host language data manipulation facilities, there is always some underlying philosophy which governs the compromises made in both design and implementation of the system, and therefore provides a basis for understanding the system.

The design philosophy of the self-contained systems is usually directed towards developing a higher-level language for use by non-programmers to provide self-contained capabilities. This philosophy usually results in several system languages such as one for interrogation and update. The specific use at the interrogation and update functions are not predefined and may change from day to day. Hence, these systems provide a mechanism for responding in a rapid manner to ad hoc interrogations and updates.

Although the systems possessing self-contained capabilities are interfaced with a general purpose operating system, they are not usually integrated with all of the facilities provided under these operating systems (e.g., they usually do not allow programming in procedural languages -- COBOL, FORTRAN within an interrogation).

The philosophy of systems with a host language capability is to provide data base management capabilities through enhancing the data definition and manipulation facilities of conventional programming languages. In some of the systems the data definition function is independent of the data manipulation functions in the host language.

GIS

GIS was developed to fill the need for a generalized system which can interrogate and maintain arbitrary user files in response to unstructured or unanticipated requests. User files are defined to the system through a special data description language. Subsequent requests for file retrieval and updating are entered in a special high-level procedural language suitable for use by non-programmers. Requests may be entered through the system input device or a terminal, and responses may be directed to the system output device or the originating terminal.

A feature is the use of storage structures which are standard to IBM's OS/360. Thus, files created by GIS may be used outside the system with a minimum of effort, and files created outside the system to its specifications may be used within the system.

MARK IV

The chief goal of the system is to facilitate batch commercial data processing by providing a simple, easy to use, interface to self-contained data base management capabilities. The use of tabular forms is intended to make basic information retrieval and maintenance functions easier for the commercial systems user who, previously, had to use a programming language. System default conditions on many form options are intended to permit users to specify a minimum amount of information to accomplish data processing tasks. As many of these tasks can be batched together as required, the system informs the user of any errors, and at all times attempts to continue processing. A design objective was to be as independent of OS and DOS releases as possible.

NIPS/FFS

This system was designed to meet the need for a system that provides responsive batch processing of large files and also handles file updating and interrogration from remote keyboard and CRT display devices. It is assumed that batch processing will be used on large files that are subject to predetermined schedules and recurring processing that is initially formulated by experienced personnel which may be initiated on a regular basis by others. The system allows the batching of many different retrievals or updates for a single pass through the file. Retrieval and report specifications are facilitated by high-level languages with English-like operators.

From remote keyboard or display devices, interrogations may be formulated and prestored interrogations or updates may be invoked by users with only a rudimentary knowledge of the system.

TDMS

The overall design philosophy was to provide a reasonably complete set of self-contained capabilities in an interactive environment. Rapid solutions are provided to a variety of user data base problems by using a language oriented to non-programming users under a time-sharing system (ADEPT-50). Further design goals include the development of a hierarchical data handling capability and the implementation of a storage structure to allow rapid response to ad hoc inquiries.

UL/1

The philosophy was to provide self-contained capabilities for a variety of non-programming users -- including those interrogating or updating a data base. UL/1 provides the capability for specifying *ad hoc* interrogations in the minimum amount of time. In addition, it provides capability for specifying computations in a specially designed procedure language embedded in it and invokable from sections of the language.

COBOL

COBOL, a procedural programming language, is not usually referred to as being a data base management system. It provides for reading, writing, and processing files contained on sequential storage devices and direct access devices. The set of data manipulation language statements provides a basic yardstick for a data manipulation language. The 1969 JOD version supplies support for enhanced character manipulation, communications oriented use, interprogram communication, and asynchronous processing in addition to the facilities of table handling, report generating, and sorting.

DBTG

The DBTG proposal provides specification of data manipulation and definition facilities to the programming user through the Data Manipulation Language (DML) and a Data Description Language. The DDL is a language for describing a total data base schema which allows data to be structured in the manner most suitable to each application, regardless of the use of the data by other applications. The schema data description provides a measure of data independence by allowing common programing languages to access the data base. The DBTG specification is directed towards multiple languages by providing a subschema (part of the total data base schema) and DML facilities specifically oriented towards the conventions of the host language.

A subschema definition capability is provided for describing that portion of the data base pertinent to a specific program or application. The subschema DDL also establishes the basis for data mapping, conversion, renaming, changing of privacy locks, and the omission of data structures not required by the program. However, the described subschema must be a logical and consistent subset of the schema from which it is drawn.

The DML extends the host language to permit selection of record occurrences, processing of data relationships, and transfer of data between the user's program and the data base. The group instance ("record") is the fundamental unit of data with which the programmer deals. The DML relies on the host language to provide required data manipulation capabilities.

DBTG explicitly recognizes the multiprogramming environment and provides DML mechanisms to preserve data base integrity.

IDS

Integrated Data Store (IDS) is a data storage and retrieval system which permits user design and implementation of a data base specifically suited to the requirements of an application. The user programs an application in a host language, COBOL, which has been augmented by IDS language elements. These elements resemble COBOL in that they are English-like and are subject to the same usage formats and conventions.

This host language augmentation provides the means for defining IDS data structures and for specifying the storage, retrieval, and update of information stored on direct access devices.

The basic unit of data is the fixed format "record" and record "chaining" is the fundamental data structuring tool. Chains are defined in the data definition function, and maintained by the system. The system was designed primarily to manipulate data records on a direct access device.

IMS

The system provides enhanced data management services called data base management services and data communications management services. They augment those provided by Operating System/360. The primary objectives of IMS are to provide an easty-to-use application programmer interface to shared data. As a consequence, some of the capabilities normally associated with data base management systems are not part of the services provided by the system.

SC-1

The system was designed to provide generalized facilities for the non-programmer, the applications programmer, and the systems programmer, all operating in a common data base environment. Facilities are provided to operate in conjunction with the vendor-supplied operating system. The programming users access and manipulate the data base using the DAMOL language which is hosted in various programming languages. Self-contained functions are provided for non-programming users for the purposes of file update and interrogation. Facilities are provided for creating the logical and physical data base environment and for tailoring that environment to particular application programs through the use of auxiliary data definitions.

1.7.2 System descriptions

A short narrative description gives a total view of the system as a whole in contrast to the individual feature analysis of the subsequent sections.

GIS

GIS is a system of generalized programs which perform the functions of file definition, file creation, file interrogation, and file maintenance. Programs are adapted to specific applications through user-supplied task specifications consisting of definitions and procedures. Task specifications are expressed in a high-level, non-programmer-oriented language.

A data base is made up of files, with each file containing a variable number of entries of a single type. Entries consist of a single master group, and optionally up to 15 levels of hierarchically related subordinate groups. A group is composed of a fixed set of fixed-size items whose types include packed decimal and EBCDIC. Items, groups, and files all have user-assigned names.

A data description task is provided for defining the data structure of the files comprising the user's data base and the input files used to update the data base. In addition to data structure, data description tasks are used to define certain aspects of storage structure; file and item access locks; input and output data validation and transformation; and event recording options. A certain amount of redefinition capability is also provided.

Procedural tasks are provided for performing the functions of file interrogation, updating, and creation. Task types include: QUERY tasks in which data from up to 16 files may be extracted for presentation in printed reports or for recording in temporary files; MODIFY tasks in which one or more "master" files may be updated from one or more "source" files; UPDATE tasks in which a single master file is updated from a single source file: and CREATE tasks in which a new file is created from a single source file. The logic of QUERY and MODIFY tasks is determined largely by the user's specifications, which may contain statements for locating and selecting instances of entries and groups, for performing arithmetic and logical operations on file data, and for generating reports and temporary files for further processing. The logic of UPDATE and CREATE tasks, on the other hand, is largely fixed by system design. The logic is that of the traditional generalized file maintenance program, and the user's specification serves mainly to supply parameters required by the generalized program.

While the primary language of GIS is a high-level, non-programmer-oriented language, certain programmer facilities are provided for extending system capabilities. These include provisions for invoking user-written assembly language routines, either explicitly from procedural task specifications or implicitly from input/output validation and conversion processes. Provision is also made for saving and subsequently invoking user-written procedural task specifications.

GIS facilities of concern to the data administration include the data definition task, the file creation and updating tasks, and a utility task for building and modifying an installation security table which controls user access to files and items within files.

GIS files are standard OS/360 data sets. Each file entry is recorded as a single data management logical record, with the groups of the entry appearing in the record in top-down left-to-right sequence. A data set may have either sequential or indexed sequential organization.

GIS operates on the System/360 under either the MFT or MVT version of Operations System/360. For operation under MFT, a configuration no smaller than a model 40 with 192K bytes of core storage is recommended, while for operation under MVT, a model 50 with 512K bytes of core storage is recommended. Under either operating system, GIS may be operated in either the batch mode or in the transaction mode. In addition, under MVT, concurrent batch and transaction processing is possible.

The transaction mode of operation requires the user to write a QTAM or TCAM (OS/360 terminal access methods) message control program which defines the user's terminal network and queues task specifications from terminals and responses by GIS. The transaction mode can thus be used from any terminal supported by QTAM or TCAM.

In the batch mode, task specifications are entered through the system input device, while in the transaction mode specifications may be entered through a terminal. Task specifications are either executed directly, or are compiled by the GIS language compiler into running programs which are then executed to carry out the tasks specified by the user. Output goes to the system output device or to the terminal where the task originated. The output consists typically of a listing of the original task specification with error indications; a listing of user-selected events (such as I/O errors) which occurred in carrying out the task; and the normal response to the task, such as a listing of selected file data.

MARK IV

The primary goal is to provide the ability to manipulate files of data. The description of these files is independent of the files themselves. The structures and format of a file and the records (entries) within that file are defined to the system and stored in a dictionary. The transactions which are used to create or update data files are likewise defined to the system and stored in a dictionary. These definitions identify the data that will update the file and specify the updating action to be performed.

After the files and their transactions have beeen defined, file maintenance can be performed. When the user specifies that a particualr file maintenance is to be invoked, the system reads the master file, reads the transactions, and does the updating.

Once files have been created, information requests can be made. These requests are used to select entries from a file, select specified data from the entries for computation and logical processing and specify the desired output. This output takes the form of reports, intermediate result files, subsets of the original file, or combinations of all of these. In addition, the system has the ability to process multiple input files simultaneously; in a single sun, the system can read five input files, generate 13 output files, and up to 255 different reports.

Requests that are to be repetitively used can be batched and stored in a system catalog as a request group; such a grouping is referred to as a cataloged job and may be subsequently invoked by specifying the request group name. In addition, each cataloged job can be modified by batching additional requests with it. For example, ad hoc requirements can be combined with a cataloged job for processing, thus alleviating the need for multiple file passes. If any requests contain errors, they will not be processed, and will not impact the processing of other valid requests.

MARK IV operates in a batch mode, serving two classes of users. The experienced user can apply the full capability of the system to accomplish his data processing tasks. The inexperienced user can use the request capability portion of the system to make ad hoc inquiries-he need only know the names of the data items and the master file they belong to.

MARK IV is intended to be used for routine batch commercial data processing. It is suited for commercial information systems (e.g., a personnel information system), and is less well suited for systems that are highly analytic, or that contain large amounts of complex arithmetic.

NIPS/FFS

This system has evolved over a number of years in a user environment. The present implementation uses a two-level hierarchical data structure with numeric, alpha numeric, and geographic coordinate data types. The user describes his data to the system by specifying item names, lengths and types, group names, conversion and editing tables and subroutines and output names. The file definition is stored in coded form at the beginning of each file in a "File Format Table." The user revises file definition by indicating changes to an existing "File Format Table." The result is a new file with the revised "File Format Table" and the applicable data from the old file. "File Format Tables" will also be provided with segments of sequential files. The segmentation of sequential files is currently being added to the system to allow users of large files to process a file segment or a concatenation of as many file segments as he needs.

Both the interrogation and update functions of the system are characterized by a multiplicity of facilities. Batch interrogations can be performed as either a one or a two step operation. In a two-step interrogation, the initial retrieval is made by the Retrieval and Sort Processor (RASP) which selects the data that meet the retrieval criteria and places it in an answer file. Answer files and/or data files may be merged and/or sorted as desired. The batch reporting processor called the Output Processor (OP) can operate on a single data file as a one-step interrogation or on a RASP answer file as the second step of a two-step interrogation. OP gives the user detailed control over report content and format.

Minimum user control over report content and format is provided by the Quick Inquiry Processor (QUIP). QUIP may be used either in batch processing or from a remote terminal. From a terminal it may be used either to invoke prestored RASP and/or OP interrogations or to specify interrogations which require a short response time and generate small amounts of output. QUIP statements are currently being expanded to give the user more power to express conditions, specify computations and control format. QUIP and OP will both be enhanced shortly by the ability to interrogate multiple files.

File updating during batch processing can be specified in three different languages representing different levels of detail in user specifications. From a remote terminal updating may be performed by invoking prestored update procedures and supplying parameters and data. Because large files make the processing of interrogations and updates a lengthy process, a restart capability is being added to the batch interrogation and the batch updating processors.

Data administration is handled at the file level by the person responsible for maintaining the file. Although NIPS/FFS provides for indicating file and report classifications, security enforcement is secured by administrative action that is external to the system.

In the near future the system will at the user's option print or record in a transaction file information on the number of times procedures are executed, amounts of core storage used, the amount of core swapping done and the number of data references included in procedures.

The storage structure used makes maximum use of the OS/360 facilities for maintaining sequential files on magnetic tape or disk and indexed sequential files on disk. These facilities will shortly be supplemented by the implementation of cross indexes within any type of file. These will be established, at the user's option, on any fixed length item. Using these facilities each entry is represented in storage by a series of similarly structured records. Each record represents the fixed group required for each entry or an instance of a repeating group.

The present implementation uses IBM S/360 model 40 or larger computers and can operate with the PCP, MFT, or MVT versions of OS/360. It uses the input/output, file access, sort, library, and program linking and loading facilities of OS/360. It also uses the Graphic Access Method and Basic Telecommunications Access Method to control remote terminals. These are supplemented by a Terminal Processor Monitor and Terminal Processor Supervisor which will soon be enhanced to allow communication from the computer operator to one or more terminal operators and from one terminal operator to one or more other terminals.

TDMS

TDMS allows users to rapidly retrieve and manipulate data from large data bases to solve pressing everyday problems. It provides this capability in a direct and uncomplicated way that does not require the user to have a sophisticated knowledge of computer technology.

With this system, the user can accomplish the following:

- Describe large collections of data and enter them into the computer using only the real characteristics of the data -- that is, name, type and relationship to other items -- without being concerned with computer functions.
- Modify data items on-line through a terminal device or off-line if the volume of his transactions warrants this kind of operation.
- Retrieve information by simply presenting queries to the computer or by requesting reports of various kinds to be generated -- again either on-line or off-line (any data item or items can be used for selective retrieval, thus freeing the user from the necessity of specifying at data definition time which items are to be indexed).
- Manage large data files -- that is, subset, merge, sort and perform other data handling functions, using a convenient language.

With this data management system, the user constructs a file by defining the data he wants to store and then presenting the actual data values to the system in the accepted format. Errors in input data are automatically presented to the user for correction so that only correct data enter the file.

If the user has a previously existing file in another format, the system provides special translation programs to achieve the necessary conversion. He is now ready to use the system to solve problems.

First, he gains direct access to the computer, typically from a remote console. He can call upon a file he has previously constructed, the names and contents of which he has determined himself- or any other file in the system that he is authorized to use. If he is not sure of how to go about solving his problem he can ask the system for help.

He can then request that selected portions of his file be presented for his analysis. He can ask questions about the characteristics of his data (e.g., limits, number of items of certain types, average values, and so on); he can change values, perform arithmetic operations on the data and combine or rearrange groups of data.

Finally, he can ask for hard-copy reports of any data--and in the format he perfers. Thus the user--that is, the person directly concerned with the problem--receives only the information he really needs, at the time he needs it and in the form most usable to him.

UL/1

The system comprises a non-procedural user-oriented language which may be used in communicating with files in a data base. The language may be used for interrogation, which involves retrieving information reports from a file and also for updating the file. This version has facilities for converting into UL/1 format from fixed field, fixed record length files. When such files are established as Ul/1 files, they may then be interrogated and updated. New files, namely files which are designed specifically for use with UL/1 may be defined having a structure with up to 15 levels of repeating groups and complete variability in item and record length.

Two levels of user are supported. The first is the data base administrator who is responsible for establishing a file in the data base. The second is a specifier user who is able to specify interrogations and updates, and is closest to the current concept of an application programmer.

A specifier user may query or update a single file in the data base. He will be able to extract information from the file in several forms. Printed reports may contain values from the file and the format of the report may be either standard or user specified. Other printed reports may be tables of the frequency of occurrence and co-occurrence of different values of specified data items. In addition to printed reports, sub-files may be extracted with either a standard record layout or with a user specified layout. In extracted files and in printed reports, sorting may be specified using any data items as sortkeys.

Interrogation may also be performed on sequential COBOL files without conversion. Such files must be definable using Spectra 70 COBOL, and the File Section of the COBOL Data Division is processed as part of the UL/1 language. It is possible to interrogate a coordinated set of up to four COBOL definable files, any or all of which may contain multiple record types (in the COBOL sense).

Updating may be specified on the record level, the data item level or on lower levels down to characters within al alphanumeric value.

Interrogation in the initial release will cause sequential searching of the file stored on either tape or disk and updating will require copying the whole file.

Files which have been established into the UL/1 format may be revised. This means that new items may be introduced into the schema for each entry and numerous other modifications may be made to the file definition data.

Finally it is possible in interrogation and update to name and to define frequently used interrogations or updates in such a way that they may be invoked several times in the same run with only the name and parameter values specified.

COBOL

The COBOL language developed out of the computer users' need for a single problem-oriented, machine independent language for business applications. While the attributes of the language make it feasible for use by application oriented personnel it is a language for programmers and is not intended to be used by people unfamiliar with the use of computers.

There are two aspects to the use of COBOL -- the language and the computer which translates the language. This feature analysis is primarily concerned with the features of the language rather than its particular implementation. The COBOL language is divided into four divisions -- IDENTIFICATION, ENVIRONMENT, DATA, and PROCEDURE.

The purpose of the Identification Division is to identify the source program and the results of the compilation process -- the output. Various attributes can be specified such as the date the program was written and the date of compilation.

The Environment Division specifies the environment in which the program is to be used. Specified in this division is the description of the hardware used to both compute and execute the program. It allows problem oriented names to be assigned to particular hardware and the physical aspect of file and storage are specified. This division is largely computer dependent.

In the Data Division the logical aspects of file and "record" (used in the object program) description are defined. The characteristics and properties of a standard data structure class are specified in the Data Division.

The Procedure Division specifies the steps that must be followed to accomplish the objectives of the program. These steps are specified in English-like imperative statements grouped into sentences and paragraphs. This division is essentially computer independent allowing the same syntax interpretation on any computer or compiler implementation.

Verbs are provided for accessing data from secondary storage, for movement (with implied editing) of data in primary storage, for arithmetic operations, and for scanning and editing strings of characters. In addition, verbs are provided for telecommunication device accessing, the searching of tables, sorting and report generation. From these,procedures can be written to perform the functions of file creation, updating,and interrogation.

DBTG

The language proposed by the DBTG includes two types of Data Description Languages (DDL) and a Data Manipulation Language (DML).

The DDL is the language used for describing a data base, or that part of a data base known to be a program. These descriptions are in terms of the names and characteristics of data-items, data-aggregates, records, areas, and sets included in the data-base, and the relationships that exist and must be maintained between occurrences of these elements in the data base.

A record is a named collection of data-items, vectors, and repeating groups which may occur an arbitrary number of times within a data base.

A set is a named collection of record types which establishes the characteristics of an arbitrary number of occurrences of the named set. A set consists of one record type declared as its owner and one or more record types declared as its member records.

An area is a named subdivision of addressable storage space in a data base which contain occurrences of records and sets. The concept of area allows the Data Administrator to subdivide a data base to control placement of records and sets, optimize concurrent access by multiple run-units, and provide a convenient unit for recovery.

A data base consists of all the record occurrences, set occurrences and areas which are controlled by a specific schema. The DDL language permits description of multiple data bases.

A schema consists of the names and descriptions of all of the areas, sets, records, and associated data-items within the data base. The DDL used to describe the schema is a language common to all sub-schema DDL's and provides a measure of data independence for program access to the data base. This capability provides support for multiple languages.

A sub-schema is a description of a portion of the data base which pertains to an individual program or application. The sub-schema DDL provides capability for data-item renaming, change of characteristics, omission of data and data structures not required, reordering and change of privacy locks for data-items and data aggregates within a record. At the record level: descriptions of specific record types may be omitted, privacy locks may be changed and access to record occurrences within areas may be controlled. At the set level: set descriptions may be omitted, privacy locks may be changed and different set selection criteria may be specified. At the area level, privacy locks may be changed and areas may be omitted. A sub-schema, however, must be a consistent and logical subset of the schema from which it is drawn. The sub-schema DDL and the DML are oriented towards the conventions of a specific host language. The DBTG proposal includes the sub-schema DDL and DML for COBOL.

The DML is the language which the programmer uses to select record occurrences, set occurrences and cause data transfer between his program and the data base. The DML relies on a host language to provide a framework for it and to provide the procedural capabilities required to manipulate data. Each DML is oriented towards the characteristics of the host language in which it is used. The DBTG DML is oriented towards COBOL.

The objectives of the DBTG language proposal is to provide features which:

- allow data to be structured in the manner most suitable to each application, regardless of the fact that some or all of that data may be used by other applications -- such flexibility to be achieved without requiring data redundancy.
- allow more than one run-unit to concurrently retrieve or update the data in the data base.
- provide and permit the use of a variety of search strategies against an entire data base or portions of a data base.
- provide protection of the data base against unauthorized access of data and from untoward interaction of programs.
- provide for centralized capability to control the physical placement of data.

- provide device independence for programs.
- allow the declaration of a variety of data structures ranging from those in which no connection exists between data-items to network structures.
- allow the user to interact with the data while being relieved of the mechanics of maintaining the structural associations which have been declared.
- allow programs to be as independent of the data as current techniques will permit.
- provide for separate descriptions of the data in the data base and of the data known to a program.
- provide for a description of the data base which is not restricted to any particular processing language.
- provide an architecture which permits the description of the data base, and the data base itself, to be interfaced by multiple processing languages.

IDS

Integrated Data Store (IDS), in its simplest sense, is a technique of data record organization which allows the application of specialized storage, retrieval, and update methods. The user defines the hierarchical relationship(s) among the data elements with which the application he is programming is concerned and specifies the methods of access to be applied. Thus IDS is a data storage and retrieval system by which data structure is tailored by the user to particular application needs.

Typically, IDS implementations are designed to operate together with COBOL as a host language. Within this symbiotic relationship, language elements relating to IDS functions describe the storage, retrieval, and update functions relating to data stored on direct access devices. The host language facilities are used to define all other data manipulation, validation, and reporting functions.

Generation of the object code for the application can be looked on as requiring two phases. In the first, the IDS langauge elements are processed to produce host compiler acceptable coding reflecting the requirements of those elements. The second phase is a standard host language compilation.

The options available to an IDS programmer are to a great extent defined by the host language and the operating system under which the generated code must operate. For example, the GECOS III Operating System for the GE 625/635 system allows concurrent read access to a common IDS file in a multiprogramming environment, while as yet there is no operating system provision for this with the GE-100 system. In other words, the IDS implementations incorporate the same basic data structuring methods. The primary differences between implementations are determined by the hardware/software system with which the user works.

An IDS file can be stored across different devices (disks and/or drums). The total space is divided into equal sized pages, in some implementations about 2,000 characters. Each physical page contains page identification and control information (a page header) and a mixture of other groups which have been defined within the IDS environment. Each group contains identification information establishing uniqueness of the group, chain pointers establishing the structural relationships between groups, and data established by the user.

The basic structural element of an IDS file is a "chain". A chain contains one "master" group and any number of "detail" groups. The master groups contains a pointer which identifies the "page and line" reference code of the first detail group. The first detail group contains a pointer which identifies the next detail group and so on until the last detail group, which points to the master group.

Data contained in a master group applies to all details chained to that master. Detail groups contain variable information pertaining to the master. If specified by the user, each detail in a chain contains additional reference pointers to identify the prior detail as well as the next detail and also the master of the chain.

A simple master/detail relationship requires definition of two group types: one master and one detail. A user may define a given IDS group as a master to more than one detail chain. In this instance, the master group contains chain references to as many separate chains of detail groups as specified. If the user wishes, he may specify a given detail group as a master to another detail group. Thus IDS allows any group to function both as master and detail to any number of other groups. The master/detail relationships among all groups within the IDS environment are specified by the user.

IDS recognizes three group classes for storage and retrieval: calculated groups, primary groups, and secondary groups.

Storage/retrieval of a calculated group is based on the value of a data item stored within the group.

Primary groups are retrieved based on a pointer furnished by the user.

Secondary groups are retrieved based on their relationship to a specified master group. Secondary groups are accessed by first retrieving the master and then stepping through the detail chain to locate the desired group.

IMS

IMS is a control program system which has been developed to facilitate the implementation of on-line data base applications. IMS is a host language system, in that the user is required to write application programs which control the transmission of messages to and from terminals and the accessing of the data base. IMS supports concurrent operation of multiple application programs. These programs may be message processing programs, conventional batch programs, or any combination of the two.

An IMS data base is composed of one or more files of tree-structured entries. Entries of the same or different files may be inter-related, permitting the non-redundant storage of data common to two entries. A data definition utility is provided for defining the structure and attributes of each file in the user's data base. In addition, an auxiliary definition facility is provided for re-defining one or more data base files as a single "logical" file. All application program access to the data base is accomplished through logical files. Logical file entries are unrelated, so the application programmer never sees structures more complex than trees.

Application programs running under IMS are written in COBOL, PL/I, or System/360 assembler language. IMS data base and telecommunication services are invoked through call statements which reference separately stored parameters. Data base services include fetching of groups having specified identifiers, fetching of the dependents of a previously obtained group, storing of new groups, and replacing and deleting existing groups. Teleprocessing services include the fetching of input messages and the transmission of output messages.

To permit recovery from hardware and software failures, IMS includes a checkpoint-restart facility and a facility for logging messages and data base modifications. Separate utility programs are provided for dumping and restoring the data base and for reconstructing a data base from the system log.

Other utility programs provided with the system include security definition, application definition, data base load and reorganization, and system log analysis programs.

Files of an IMS data base may have one of two organizations: hierarchical sequential (HS) and hierarchical direct (HD). The HS organization has two access methods, HSAM and HISAM, which are based on the OS/360 sequential access method (SAM) and indexed sequential access method (ISAM), respectively. HISAM provides a single index for accessing file entries on the basis of a user-declared entry identifier.

The HD organization employs a unique block storage technique, in which one or more groups are stored in physical blocks of the same size, and relationships between groups (hierarchic and sibling) are maintained through relative block addresses imbedded in the blocks. The HD organization also provides two access methods: HDAM in which entries are accessed through a hash-addressing technique, and HIDAM in which entries are accessed through an index as in HISAM.

The IMS control program runs in a single region under OS/360 MVT or MFT. One or more additional regions are used for user message processing programs and batch programs accessing the data base. The control program analyzes incoming messages, queues them, and dispatches the appropriate application programs. The application program retrieves its message(s) from the queue, and carries out its processing, which typically includes data base access and transmitting a response to the sending terminal. Additional functions performed by the IMS control program include message switching, message editing, and interpretation of master and user terminal commands for effecting various kinds of system control.

SC-1

SC-1 is a data management system that operates with a common data base. It includes self-contained functions for updating and interrogating the data base as well as the data manipulation language facilities necessary for the support of host-language applications.

System jobs or utilities are used to create the data base environment. Users employ special-purpose languages to perform such functions as system data definition, data location, data security definition, restructure or redefinition of the data base, and the specification of auxiliary data definitions.

User programs written in conventional programming languages are served by a set of data manipulation facilities called the Data Services. These services interpret a program's request for data or data manipulation. The requests are made via a data management language (DAMOL) statement which is embedded in the user's program. This statement gives the specifications necessary for the Data Services to interact with the operating system to perform the requested operation.

The SC-1 system contains a generalized input package called Message Discrimination and Validation, File Update and Surveillance (MDV/FUS) for introducing external data into the system to update the data base. Users of this package employ tabular forms and a special purpose narrative language called MPL (Message Processing Language) to describe the types of input (the unit of input is called a message), the validation checks to be performed on this input, and the data base updating to be performed using these input messages. These user-defined specifications serve as input to a system job called Page Builder which stores them in "pages" in the data base for reference during a processing application. Users have the option of using the MDV/FUS input facility in one of two ways. He can use the Standard Mode of operation, submitting batches of input messages of different types to the MDV function which performs the appropriate validity checks and stores the validated messages for processing at a later time via a FUS job, or he can use the Pipeline Mode in which an individual message is checked for validity (MDV) and processed against the data base (FUS) before the next input message is identified.

The generalized output or interrogation package is called Report Data Compiler (RDC). It also uses a special purpose narrative language through which users may specify the structure, content, and format of desired output reports. As in MDV/FUS the report specifications are compiled and stored for reference during a processing application. The RDC facility contains the additional feature of being able to invoke pre-stored tables or subroutines coded in a conventional programming language.

Both MDV/FUS and RDC system programs use the Data Services (via DAMOL) to interact with the data base. These functions are designed to accommodate the bulk of input/output processing applications by non-programmers.

1.8 Selected bibliography

GIS

Technical articles:

Bryant, J. and P. Semple, "GIS and File Management," Proc. ACM 21st National Conference, 1966, pp. 97-107.

Manuals:

GIS (Basic) Application Description Manual H20-0521
GIS Application Description Manual H20-0574

MARK IV

Technical Articles:

Postley, J.A., "The MARK IV System," Datamation, January 1968, pp. 28-30.

Manuals:

MARK IV Reference Manual, Document No. SP-68-810-1.

NIPS/FFS

Manuals:

The National Military Command System Information Processing System 360 Formated File System (NIPS 360 FFS), Users Manual, Computer System Manual Number CSM UM 15A-68, 15 January 1970 consists of the following:

Vol I	Introduction to File Concepts, changes 1 and 2
Vol II	File Structuring (FS), changes 1 and 2
Vol III	File Maintenance (FM), changes 1 and 2
Vol IV	Retrieval and Sort Processor (RASP), changes 1 and 2
Vol V	Output Processor (OP), changes 1 and 2
Vol VI	Terminal Processing Component (TP), change 1
Vol VII	Utility Support (UT), changes 1 and 2
Vol VIII	Job Preparation Procedures, changes 1 and 2
Vol IX	Error Codes, change 1
TR 54-70	Installation of NIPS 360 FFS, changes 1 and 2

TDMS

Technical Articles:

SP-2747, "The Time-Shared Data Management System: A New Approach to Data Management."

SP-2634, "COMPOSE/PRODUCE: A User-Oriented Report Generator Capability Within the SDC Time-Shared Data Management System."

SP-2750, "Treating Hierarchical Data Structures in the SDC Time-Shared Data Management System (TDMS)."

SP-2907, "File Organization in the SDC Time-Shared Data Management System (TDMS)."

Manuals:

TM-3370, The Time-Shared Data Management System (TDMS) Language Specifications

TM-3849, The Time-Shared Data Management System Interim User's Guides

UL/1

Technical Articles

Olle, T.W., "UL/1: A Non-Procedural Language for Retrieving Information from Data Bases," *Proc. IFIP Congress 1968*, Edinburgh, Scotland.

Olle, T.W., "UL/1: A User Language also for Document Applications." Proceedings of ASIS Conference 1968, pp. 151-153.

Manuals:

RCA Computer Systems TDOS UL/1 Version 2 Reference Manual, May 1971.

COBOL

"CODASYL COBOL Journal of Development 1969",Number 110-GP-1a, April 1970. Available from The Canadian Government Specifications Board, Ottawa, Canada.

DBTG

Technical Articles:

"CODASYL Data Base Task Group," April 1971, Report.

IDS

Technical Articles:

Bachman, C.W. and S.B. Williams, "A General Purpose Programming System for Random Access Memories," Proceedings Fall Joint Computer Conference 1964, pp. 411-422.

Bachman, C.W., "Software for Random Access Processing," Datamation, April 1965, pp. 36-41.

Manuals:

Integrated Data Store - a New Concept in Data Management, Publication CPB-483, General Electric Company Information Systems Department, Phoenix, Arizona.

Integrated Data Store Base Study, Bachman, C.W., Publication CPB-481, General Electric Company Information Systems Department, Phoenix, Arizona.

Introduction to Integrated Data Store, Publication CPB-1048, April 1965, General Electric Company Information Systems Department, Phoenix, Arizona.

GE 625/635 Integrated Data Store (GECOS III), Publication CPB-1565, December 1968, General Electric Company Information Systems Department, Phoenix, Arizona.

IMS

Manuals:

Information Management System/360, Program Description	SH 20 0634-1
General Information Manual	GH 20-0765
System/Application Design Guide	SH 20-0910
Application Programming Reference Manual	SH 20-0912
System Programming Reference Manual	SH 20-0911
Operator's Reference Manual	SH 20-0913
Utilities Reference Manual	SH 20-0915

SC-1

Technical Articles:

Welsh, W.A., "Engineered Design of EDP Systems," *Systems and Procedures Association International Meeting*, October, 1968.

Nilson, N.W., "The Logical Data Base - Its Concepts, Development and Use," *AMA Conference*, July 9, 1968.

Welsh, W.S., "Engineering a Computer-Based Information System," *AMA Conference Planning and Implementing a Computer-Based Management Information System*, September 1967.

The following Auerbach documents describe the DM-1 concept used in development of SC-1:

Sable, J.E., *et al.*, "Design of Reliability Central Data Management Subsystem." Final Report. July 1965, Vol. 2. (RADC TR-65-189 Vol. 2) (AS-469-269).

Sable, J., W. Crowley, H. Rosenthal, S. Forst, and M. Harper, "Reliability Central Automatic Data Processing Subsystem. Vol. 1: Design Specification Report." Final. August 1966, 131 p. (report No. 1280-TR-Vol. 1) RADC TR-66-474-Vol. 1 (AD-489-666).

Dixon, P. and J. Sable, "DM-1 -- A Generalized Data Management System," Proceedings Spring Joint Computer Conference 1967, p. 185.

Manuals:

System Control-1, User's Guide. Information Systems Engineering, Western Electric, 80 Mulberry Street, Newark, New Jersey 07102, 1970, January.

2. DATA STRUCTURES

Most data base systems permit users to interact with the data in terms which are independent of the manner in which the data are physically stored. In such systems, the user's conception of the data is called a data structure. In contrast, the data collection as physically stored is called a storage structure.

This chapter describes the class of data structures which systems make available to the user. Storage structure aspects are covered in Chapter 9. Wherever data structure and storage structure facilities are indistinguishable, the facilities are covered here rather than in Chapter 9.

The data structures of primary interest in this section are those comprising the user's data base, i.e., the structures which are created and maintained through user-specified creation and update processes, and from which reports and other structures are derived through user-specified interrogation processes. Such processes may make use of ancillary data structures which are not normally considered part of the data base; for example, updating may make use of transaction files, interrogation may produce answer files. These ancillary structures are described in connection with the process which creates or uses them.

A data base is composed of structures or elements of different types, with structures of one type being composed or constructed from structures of other types. For example, a certain system might provide structures of three types: "fields", "records", and "files". "Records" are constructed from "fields", and "files" are constructed from "records".

Each type of structure has a set of attributes which distinguish it from other structure types. In the example of the preceding paragraph, attributes of "fields" might be name, length, value, and date of last change in value. Similarly, attributes of "records" might be name and date of addition to the file.

The data structure class of a system can be described by identifying the types of structures it provides, and for each structure type specifying

- the manner in which structures of this type are constructed from other structures, and
- the attributes of structures of this type and the values which these attributes may assume.

The description of a data structure class is simplified by postulating a fixed set of generic structure types, and by equating the structure types of a given system with these generic types. The following generic structure types are used for this purpose:

Item

Group

Group relation

Entry

File

Data Base

The general manner in which each of these structure types is used in composing structures of higher order is shown in Figure 2-1. The types are further defined in the sequel.

In addition to the generic structure types listed in the preceding paragraph, systems frequently provide multiple structure types within a generic type. For example, a system may provide several types of "fields", several types of "records", and so forth. In general, two structures can be considered to be of different type if they have different sets of attributes, or if they are generated and manipulated according to different sets of rules.

The description of a data structure class is further simplified by introducing the concept of a structure schema. A schema is a description or definition of a set of structures of a given type in terms of a certain subset of the attributes for that type. The attributes selected for this purpose are called the schema attributes. In the previous example, the schema attributes for the structure type "field" might be name and length. The schema for a set of "fields" would consist of a value for each of these schema attributes, e.g.,

STRUCTURE	COMPONENT STRUCTURES
item	none
group	item, group
group relation	group
entry	group, group relation
file	entry, group relation
data base	file, group relation

Figure 2-1
Inter-relation of generic structure types

name	length
EMPNAME	15

implying that each "field" in the set has the name EMPNAME and a value whose length is 15 characters.

The structures defined or described by a schema are sometimes referred to as instances of the schema. The following are instances of the schema in the preceding paragraph:

name	length	value	date of last change in value
EMPNAME	15	'N. M. bPETERSONƀƀ'	700207
EMPNAME	15	'K. R. ƀLYNNƀƀƀƀƀƀ'	701212
EMPNAME	15	'J. ƀFINSTERWALDƀ'	690305

In general, the value of a non-schema attribute will vary from one instance of the schema to another.

The notion of a schema simplifies the description of a data structure class by permitting the composition rules for a given structure type to be broken into two parts:

1) rules for composing a schema; and
2) rules for generating instances of a schema (sometimes called "populating the schema").

Similarly, the attributes of a structure type can be conveniently divided into schema attributes and non-schema attributes.

Several of the systems surveyed (DBTG, IMS, and SC-1) provide two distinct data structure classes, one for the data administrator and one for the application programmer or self-contained facility user. In the sequel the data structure classes of these systems are distinguished when necessary by appending DA (for data administrator) or AP (for application programmer) to the system acronym. For example, SC-1 (AP) refers to the SC-1 application programmer data structure class.

The description of an AP data structure class includes a description of constraints placed by the system on the derivation of AP data structures from DA data structures.

DBTG is intended eventually to have a number of AP data structure classes. The only one defined to date is a class of structures intended for the COBOL programmer. Except in minor respects, this class is the same as the COBOL data structure class described herein. Accordingly, no attempt has been made in the sequel to describe DBTG (AP). The class described for DBTG is actually DBTG (DA), i.e., the DBTG data administrator's data structure class.

2.1 Items

The item is the elementary data structure from which structures of all other types are ultimately composed. The principal attribute of the item is value, which may be a number, a string of characters, a truth value, and so on. Other attributes might include a name which is used to refer to the item, a value existence indicator to indicate the presence or absence of a valid value, an access lock to control access to the item, and so forth.

In data base applications, items are usually associated with the attributes of an application entity. For example, the entity person might have attributes name, age, and sex. By associating an item with each of these attributes, the value of the item can be used to represent the value of the associated entity attribute.

All systems surveyed possess the item level of data structure. Figure 2-2 indicates the terms used by the various systems for items.

In IDS, items are defined but the definitions are in effect immediately turned over to the COBOL compiler; IDS itself keeps no information on item attributes.

2.1.1 Item types

Most systems provide several types of items. The different types are distinguished primarily by the kind of values which items of each type may have. For example, items of type numeric might be restricted to values which are numbers, items of type alphabetic to values which are strings of letters, and so on. In general each item type will have its own characteristic set of attributes.

Multiple item types are generally provided to permit a more natural representation of application entity attribute values. For example, it is more natural (and possibly more efficient) to associate the entity attributes name and age with items of different types (e.g., alphabetic and numeric, respectively) than to associate them with items of a single type.

GIS	field
MARK IV	field
NIPS/FFS	field
TDMS	element
UL/1	item
COBOL	elementary item
DBTG	data item
IDS	data field
IMS	field
SC-1	field

Figure 2-2
System's term for item

Item types can be classified into three categories: numeric, string, and other.

All systems possess multiple item types, as indicated in the following sections under the appropriate category.

By IMS, item type may be declared by the user, but this information is not used by the system. The system treats all items as byte strings.

2.1.1.1 Numeric item types

A numeric item is one whose value is a number and which can be used in arithmetic operations.

All systems possess at least one numeric item type, and a number of systems provide several. The different numeric types in a given system usually reflect different storage representations. For example, in MARK IV, the "packed decimal" and "zoned decimal" item types employ the IBM System/360 packed decimal and zoned decimal representations, respectively. In such cases, the user is able to control item representation through declaration of item type, but at the same time must be aware of any system restrictions which may exist on the combining of numeric items of different types. In contrast, in systems like UL/1, the user has only a single numeric type at his disposal; the system determines the appropriate representation for each item on the basis of the value initially supplied, and takes care of any necessary subsequent conversions between representations.

Figure 2-3 lists the numeric item types available in each system and gives the range of numbers accommodated by each. Ranges expressed in bytes allude to the storage representation used for these item types. The notation "n bytes fixed" signifies an (8n)-bit two's complement binary number having a range of -2^{8n-1} to $+2^{8n-1}-1$. The notation "n bytes floating" signifies a normalized hexadecimal floating point number in System/360 format, with a range of approximately -10^{75} to 10^{75}. For n=4 the precision is equivalent to approximately 7 decimal digits and for n=8 the precision is equivalent to approximately 16 decimal digits.

	SYSTEM TERM	VALUE RANGE
GIS	packed decimal	1 to 31 digits, signed
	right-justified EBCDIC	1 to 31 digits, signed
MARK IV	packed decimal	1 to 15 digits, signed
	zoned decimal	1 to 15 digits, signed
	fixed binary	1 to 4 bytes fixed
	floating binary	4 bytes floating
NIPS/FFS	numeric	1 to 10 digits, signed
TDMS	number	1 to 255 characters representing signed or unsigned integers, decimal numbers, or numbers in scientific notation.
UL/1	numeric	2 bytes fixed, 4 or 8 bytes floating
COBOL	numeric	1 to 18 characters representing signed or unsigned integers or decimal numbers
DBTG	arithmetic[1]	Value range is implementor-defined
IDS	numeric	1 or more characters representing signed or unsigned integers or decimal numbers
IMS	hexadecimal	Item representation limited to 255 bytes
SC-1	decimal integer	1 to 31 digits, signed
	binary integer	1 to 8 bytes, fixed
	exponential	4 or 8 bytes, floating

[1] See 2.1.2.3 for numeric item "subtypes".

Figure 2-3
Numeric item types

2.1.1.2 String item types

A string item is one whose value is a sequence of characters from a finite alphabet. String items may be used to represent numeric entity attributes, but are more typically used to represent alphabetic or alphanumeric attributes.

All systems provide at least one string item type. NIPS/FFS has two string item types: one for fixed length items, the other for variable length items. COBOL, DBTG, and IDS also have two: one for items whose values consist of alphabetic characters only, the other for items whose values are composed of any character allowed in the system.

Figure 2-4 lists the string item types available in each system and gives the length of strings accommodated by each. The figure 255 stems from storage structure considerations. The designation EBCDIC stands for Extended Binary Coded Decimal Interchange Code, an 8-bit character coding scheme used in System/360.

2.1.1.3 Other item types

Item types not properly classified as numeric or string include "date", "coordinate", and "Boolean".

GIS

Item types "binary" and "floating point" are provided to serve as "filler" in the containing group. Binary items may have lengths of 1 to 4 bytes, and floating point items may have lengths of 4 or 8 bytes. These types are provided as a user convenience in describing existing files containing System/ 360-generated binary and floating point data. GIS itself does not perform binary or floating point operations, but instead treats all items of types "binary" and "floating point" as if they were of type "left-justified EBCDIC" (a string item type).

MARK IV

None.

NIPS/FFS

A "geographic coordinate" item type is provided for expressing geographic latitude and longitude in degrees, minutes and (optionally) seconds.

	SYSTEM TERM	VALUE RANGE
GIS	left-justified EBCDIC	1 to 255 EBCDIC characters
MARK IV	EBCDIC	1 to 255 EBCDIC characters
NIPS/FFS	alphameric variable length	1 or more EBCDIC characters[1] 1 or more EBCDIC characters[1]
TDMS	name	1 to 255 characters
UL/1	alphanumeric	0 or more characters (entry-level restriction)
COBOL	alphabetic alphanumeric	1 or more alphabetic characters and blanks 1 or more alphanumeric characters
DBTG	character bit	1 or more characters 1 or more bits
IDS	alphabetic alphanumeric	1 or more alphabetic characters and blanks 1 or more alphanumeric characters
IMS	alphanumeric	1 to 255 characters
SC-1	alphanumeric bit string general	1 to 255 EBCDIC characters 1 to 2040 bits 1 to 254 bytes

[1] Length limited only by storage structure.

Figure 2-4
String item types

TDMS

A "date" item type is provided for calendar dates of the form mm/dd/yy where mm = month (1 to 12), dd = day (1 to 31), and yy = year (1 to 99).

UL/1

A "date" item type is provided for calendar dates in the forms <year, month, day>, <year, day of year>, or <year>.

A "coded" item type is provided for items whose values, consisting of one to 8 non-blank characters, are to be transformed on output into strings of one or more characters, through a user supplied decoding mechanism.

COBOL

None.

DBTG

A "database-key" item type is provided to hold system-supplied identifiers for "record" instances (see 2.2.3.3).

IDS

None.

IMS

None.

SC-1

None.

2.1.2 Item schema attributes

2.1.2.1 Names

Systems generally provide facilities for assigning names, numbers, or other identifiers to items. These facilities may include assigning synonyms and external names, such as column headings in reports.

All of the systems surveyed require the user to declare a name for each item schema. All systems provide for assigning alphanumeric names, and two systems (TDMS and UL/1) also require user-declared item numbers.

The systems differ with regard to the requirement for item name uniqueness. In some systems, item names need only be unique within the containing group. Since the same name may be used for a different item in another group, some form of qualification must be provided for referring unambiguously to a given item. In COBOL, for example, item references may be qualifified by appending the phrase "IN group-name" to the item name.

Five systems (GIS, MARK IV, COBOL, IDS, and IMS) make specific provision for item synonyms, i.e., names which may be used interchangeably with the first item name. In a number of other systems, an item can indirectly be given a synonym by defining a group comprised of the single item, and referring to the item by the group name. However, this is not the principal intent of this facility, and strictly speaking is not a synonym facility.

Except for TDMS, all of the self-contained systems provide for a separate user-supplied output heading for each item. In TDMS, item names (which may be as long as 255 characters) are used to identify output values. None of the host-language systems provides for output headings.

	NAME TYPE		UNIQUE WITHIN	SYNONYMS	OUTPUT HEADING
	ALPHANUMERIC	NUMBER			
GIS	1 to 8 char.	none	group	yes	1 to 132 char.
MARK IV	1 to 8 char.	none	entry	yes[1]	1 to 126 char.
NIPS/FFS	1 to 7 char.	none	entry	no	1 to 69 char.[2]
TDMS	1 to 255 char.	1-32755	entry	no	no
UL/1	1 to 8 char.	1-255	entry	no	1 to 160 char.
COBOL	1 to 30 char.	none	group	yes[3]	no
DBTG	1 to 30 char.	none	group	no	no
IDS	1 to 30 char.	none	group	yes	no
IMS	1 to 8 char.	none	group	yes[1]	no
SC-1(DA)	2 to 31 char.	none	group	no	no
SC-1(AP)	2 to 31 char.	none	entry	no	no

[1] Through item redefinition.

[2] In the terminal processor (QUIP) only.

[3] For fixed length items in non-repeating groups only.

Figure 2-5
Item naming

2.1.2.2 Value class attributes

The value class of an item is the set of values that may be assigned to the item. For example, the value class of an item named SEX might be the set {'M', 'F'}, implying that the value of SEX must be either 'M' or 'F'.

The value class of an item is determined primarily by the item's type. In addition, the item may have one or more value class attributes which serve to further constrain the set of values which the item may assume. For example, a string item might have the attribute length, specifying the number of characters required in item value. The item might also have the picture attribute which specifies not only the permissible length of an item value but also the characters permitted at each position in the value.

Typical value class attributes are:

- length or length range of string item values
- picture of string item values
- precision of numeric item values (for example, number of decimal places)
- range of numeric item values
- lists of discrete item values
- expressions which may be evaluated in specific context to yield a value, such as the expression
  LENGTH IS ITEMLENGTH
  which becomes, in the context of ITEMLENGTH=6,
  LENGTH IS 6

Value class attributes are used to validate values being supplied for data base items. The mechanics of doing this are described in Chapters 5 and 6.

Figure 2-6 lists the common value class attributes of the numeric item types in the systems surveyed. The length attributes fixed length (L) and length range (R) are typically expressed in storage structure units - for example, bytes - and to this extent the user is aware of the representation used for numeric items. A fixed length attribute implies that the values in all instances of the item schema have the same length. A length range attribute implies that the value length may vary between user-specified limits - typically zero and some maximum number. No length attribute also implies that the value length is variable, but with limits on the variability being determined by the system.

	ITEM TYPE	LENGTH[1] ATTRIBUTES	PICTURE	DECIMAL LOCATION	VALUE RANGE	VALUE LIST	VALIDATION ROUTINE
GIS	packed decimal	L	yes	no	yes	yes	yes
	right-justified EBCDIC	L	yes	no	yes	yes	yes
MARK IV	packed decimal	L	no	yes	no	no	no
	zoned decimal	L	no	yes	no	no	no
	fixed binary	L	no	yes	no	no	no
	floating binary	L	no	yes	no	no	no
NIPS/FFS	numeric	L	no	no	no	no	yes
TDMS	number	none	yes	no	yes	yes	no
UL/1	numeric	none	no	no	yes	yes	no
COBOL	numeric	L or R	yes	yes	yes	no	no
DBTG	arithmetic	L or none	yes	yes	yes	no	yes
IDS	numeric	L	yes	yes	yes	no	no
IMS	hexadecimal	L	no	no	no	no	no
	packed decimal	L	no	no	no	no	no
SC-1 (DA)	decimal	L or R	no	no	no	no	no
	integer	L or R	no	no	no	no	no
	exponential	L	no	no	no	no	no
SC-1 (AP)	decimal	L	no	no	no	no	no
	integer	L	no	no	no	no	no
	exponential	L	no	no	no	no	no

[1] L=fixed length
R=length range

Figure 2-6
Numeric item value class attributes

Figure 2-7 gives similar value class attributes for the system's string item types. For string item types, the length attributes are expressed in data structure units - e.g., characters - which in many cases happen to coincide with storage units, e.g., one character equals one byte.

Other value class attributes not shown in the figures are as follows:

UL-1

The value class attributes shown in the tables are specified through the same language that is used to specify conditional expression (see Chapter 4). Thus, value class attributes considerably more complex than these are possible.

For items of type "coded", the value list attribute is provided.

SC-1 (DA)

An "optional value" attribute is provided for indicating whether a value may be missing in an instance of a fixed length item schema.

2.1.2.3 Other item schema attributes

Other possible schema attributes for an item include:

- the units in which values are expressed (feet, pounds, dollars, etc.)
- usage of item values (for example, "computational" or "display")
- output editing attributes such as edit masks
- input and output conversion attributes
- value synonym lists (such as MALE for 'M', FEMALE for 'F')
- subitem schemas (a subitem is a string type item whose value is a substring of another string type item)
- access locks for controlling access to item values

Figure 2-8 lists other item schema attributes provided in the systems surveyed. Not included in the figure are the following attributes:

COBOL

The "justified" attribute is used to specify non-standard justification of string-type item values.

	ITEM TYPE	LENGTH ATTRIBUTES[1]	PICTURE	VALUE RANGE	VALUE LIST	VALIDATION ROUTINE
GIS	left-justified EBCDIC	L	yes	yes	yes	yes
MARK IV	EBCDIC	L or R	no	no	no	no
NIPS/FFS	alphameric	L	no	no	no	yes
	variable length	none	no	no	no	no
TDMS	name	none	yes	yes	yes	no
UL/1	alphanumeric	L or R	yes	yes	yes	no
COBOL	alphabetic	L or R	yes	yes	no	no
	alphanumeric	L or R	yes	yes	no	no
DBTG	character	L or none	yes	yes	no	yes
	bit	L or none	yes	yes	no	yes
IDS	alphabetic	L	yes	yes	no	no
	alphanumeric	L	yes	yes	no	no
IMS	alphanumeric	L	no	no	no	no
SC-1 (DA)	alphanumeric	L or R	no	no	no	no
	bit-string	L	no	no	no	no
	general	L or R	no	no	no	no
SC-1 (AP)	alphanumeric	L	no	no	no	no
	bit-string	L	no	no	no	no
	general	L	no	no	no	no

[1] L=fixed length
R=length range

Figure 2-7
String item value class attributes

	SUBITEMS	OUTPUT EDIT MASK	INPUT/OUTPUT CONVERSION	ACCESS LOCKS
GIS	yes	yes	routine,table	query, update
MARK IV	no	yes	table	none
NIPS/FFS	no	yes	routine, table	none
TDMS	yes	no	none	none
UL/1	yes[1]	no	table[2]	none
COBOL	no	yes	none	none
DBTG	no	no	yes	query, store, modify
IDS	no	yes	none	none
IMS	no	no	none	none
SC-1	no	no	none	access, query, update

[1] For item type "date" only.
[2] Output conversion only.

Table 2-8
Other item schema attributes

One or more "condition-names" may be associated with an item schema. Associated with each condition-name is a set of values or value-ranges for which the condition-name may be used as a synonym. For example, for the item AGE, the condition-name YOUNG might be defined for values less than 40 and the condition-name OLD for values over 40.

An item is multiple-valued (see 2.1.3) if it has the OCCURS attribute. The OCCURS attribute specifies the number of instances of the schema that can occur within a given parent group. This number may be a constant or a variable. Multiple-valued items must be fixed length.

A multiple-valued item schema may have the same attributes as a repeating group schema, namely, a "sequencer" (which is the item itself) and an "index" (see 2.2.3.2).

DBTG

An item schema may have the OCCURS attribute as in COBOL. Unlike COBOL, multiple-valued items need not be fixed length.

For "numeric" item types, one attribute from each of the following sets may be associated with an item:

$$\{\text{binary, decimal}\}$$
$$\{\text{fixed, floating}\}$$
$$\{\text{real, complex}\}$$

The result is eight numeric subtypes (e.g., "binary fixed real").

A "source" attribute specifies that all attributes of an item except name are to be the same as the attributes of a specified "source" item. The source item must occur in a parent "record". The value of an item with the "source" attribute cannot be changed independently of its source item.

A "result" attribute specifies that the value of an item is to be derived by executing a user-supplied procedure. If the attribute is "actual result", the item value is computed whenever a change occurs in the value of any item in the procedure. If the attribute is "virtual result", the item value is computed whenever the item is referred to. Items referred to in the procedure must occur in the same group as the derived item, or in a dependent assembly of this group.

IMS

The relative position ("displacement") of the item schema in the group schema is a data structure attribute.

SC-1

Two levels of access locks are provided: a "security restriction level" which controls access of any kind; and "user access class" and "user modification class" for controlling interrogation and update, respectively.

2.1.3 Other item attributes

The principal non-schema attribute of an item is value. Other non-schema attributes include:

- indicators for indicating non-existence of an item value (for example, because it is not known or it has been deleted)
- date and time of last change to the item value
- identification of transaction or program which supplied the current item value

In addition, any attributes listed in 2.1.2 which in a given system are allowed to vary from one instance of an item schema to another would be non-schema attributes.

The non-schema attributes of an item are normally single-valued. However, some systems provide for multiple-valued items, i.e., items whose non-schema attributes may be multiple-valued. In particular, the value attribute of the item may have multiple values. An instance of a multiple-valued item schema consists of n values for each of the non-schema attributes of the item, where n may be zero or more. Such an instance differs from n instances of a single-valued item schema, in that the former may occur in a single group, whereas the latter must occur in n different groups.

In addition to value, the following non-schema attributes are provided in the systems surveyed:

GIS

Null item values are provided.

MARK IV

None.

NIPS/FFS

None.

TDMS

Null item values are provided.

UL/1

Null item values and multiple-valued items are provided. An optional date stamp may be associated with each item.

COBOL

Multiple-valued items (fixed length only) are provided.

DBTG

Multiple-valued items are provided.

IDS

None.

IMS

None.

SC-1

Null item values are provided for fixed-length items. For variable-length items, a length of zero signifies a null value.

A system identifier, the "item position code", is associated with each item, and may be used to access items.

2.2 *Groups*

A *group* is a set of items and possibly other groups. A group composed solely of items is called a *simple group*; a group containing both items and groups is called a *compound group*.

A *simple group* is a way of collecting together a set of items and giving the set a name and other attributes of its own. The group may correspond to an application entity, with the items of the group corresponding to application entity attributes. For example, the group PERSON composed of items NAME, NUMBER and SALARY might correspond to an employee with the attributes *name*, *number*, and *salary*.

A simple group is also a way of referring to the "collective value" of a set of string-type items. The collective value can be defined as the concatenation of the constituent item values in some prescribed order. Thus, if the string items YEAR, MONTH, and DAY have values "70", "12", and "25", respectively, the group DATE = {YEAR, MONTH, DAY} would have the value "701225". The grouping of string items achieves the same effect as the partitioning of a string item into subitems (see 2.1.2.3), and in some systems no distinction is made between the two kinds of facilities.

A simple group schema consists of zero or more distinct item schemas. Such a schema can be portrayed by a tree-like diagram in which the root stands for the group and the leaves stand for the items. Figure 2-9 illustrates the schema for the group PERSON.

An empty group schema is useful for collecting under a single parent a set of group schemas having no items in common.

An instance of a simple group schema contains one instance of each constituent item schema. The instance can also be portrayed as a tree, except that in this case the leaves stand for item instances. Figure 2-10 shows two instances of the group schema PERSON.

A *compound group* provides a way of collecting together a set of items and a set of groups, and giving the new set a name and other attributes of its own. A reference to a compound group is a reference not only to its immediate constituent items, but to the items in all of its included groups as well. Thus, if the group SKILL = {CODE, TITLE} were added to the simple group PERSON = {NAME, NUMBER, SALARY}, the result would be the compound group

PERSON = {NAME, NUMBER, SALARY, SKILL = {CODE, TITLE}}.

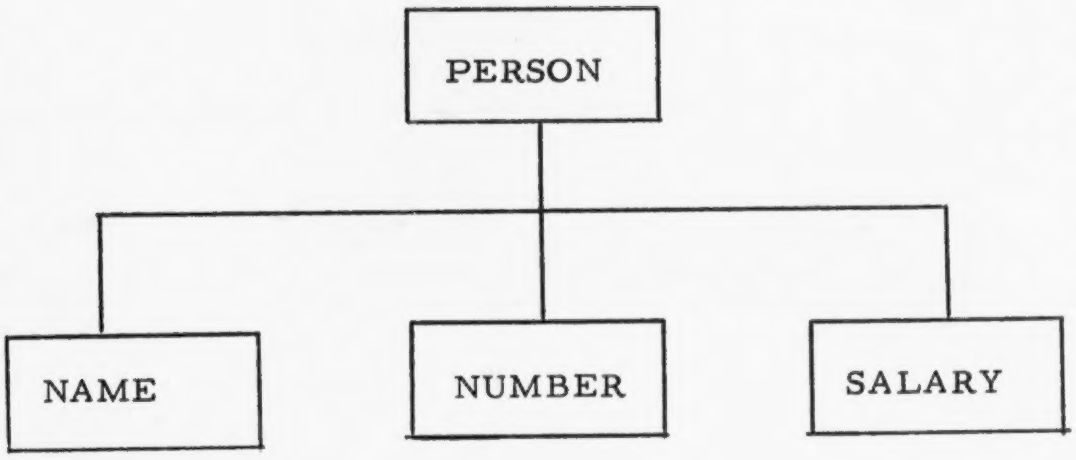

PERSON
NAME
NUMBER
SALARY

Figure 2-9
A simple group schema

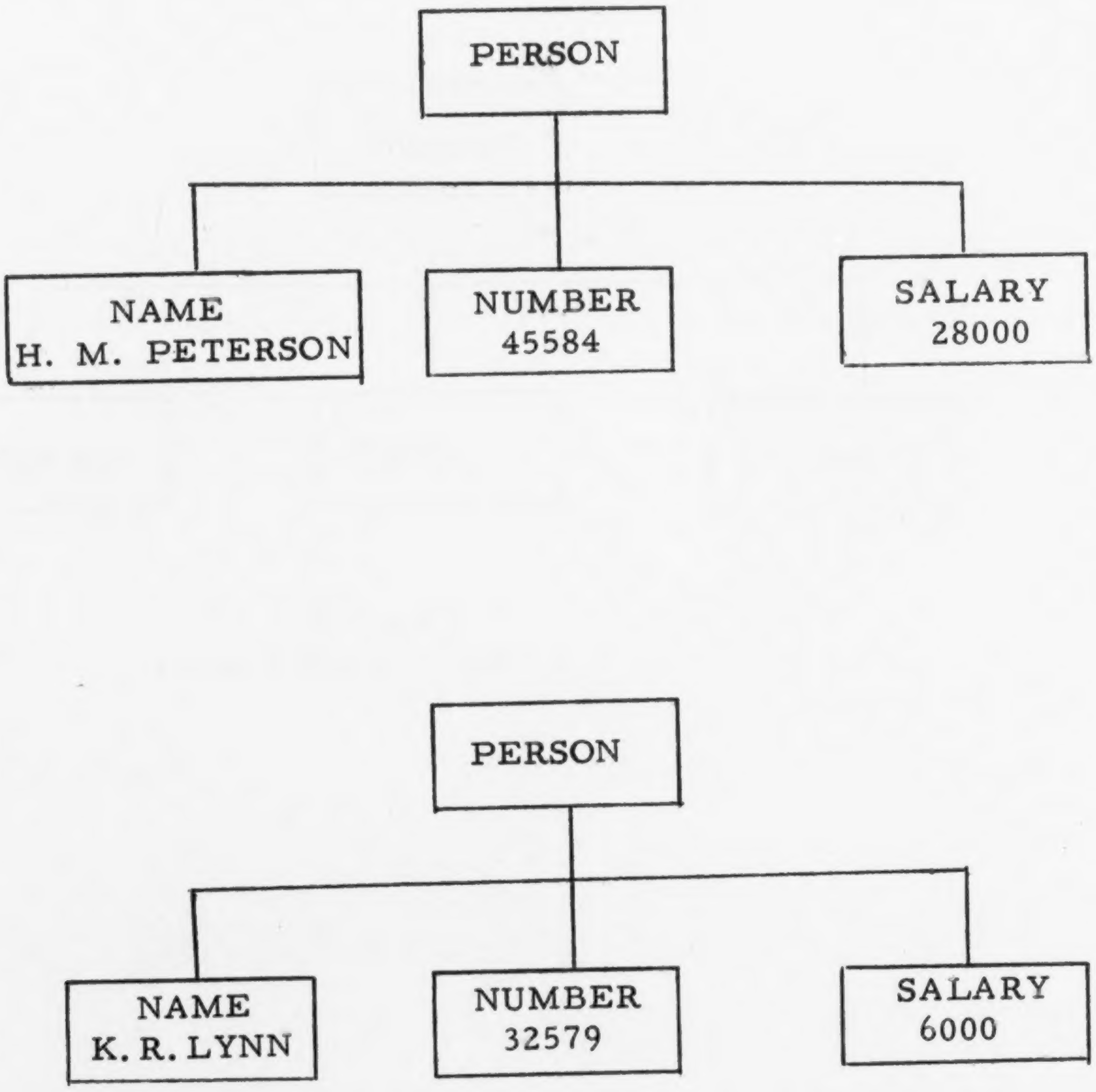

PERSON
NAME
H. M. PETERSON
NUMBER
45584
SALARY
28000
PERSON
NAME
K. R. LYNN
NUMBER
32579
SALARY
6000

Figure 2-10
Two instances of a simple group schema

A reference to PERSON would now be equivalent to a reference to NAME, NUMBER, SALARY, CODE, and TITLE.

The items which are the immediate constituents of a compound group are called principal items. Similarly, the groups which are the immediate constituents of the compound group are called principal groups. In the compound group

$$\text{PERSON} = \left\{\text{NAME, NUMBER, SALARY, SKILL} = \left\{\text{CODE, TITLE}\right\}\right\},$$

the items NAME, NUMBER, and SALARY are principal items, and the group SKILL is a principal group.

A compound group also provides a way of establishing a hierarchic relation between two sets of items, hence between the application entities to which the sets of items correspond. In particular, the items in a principal group of a compound group stand in a subordinate relation to the principal items in that compound group. Thus, in the compound group

$$\text{PERSON} = \left\{\text{NAME, NUMBER, SALARY, SKILL} = \left\{\text{CODE, TITLE}\right\}\right\},$$

the items CODE and TITLE (representing the entity skill) are subordinate to the items NAME, NUMBER and SALARY (representing the entity person).

A compound group schema consists of zero or more distinct item schemas and one or more distinct group schemas. A principal group schema may be either simple or compound. Thus, groups may be "nested" to any depth in a compound group. Figure 2-11 illustrates the schema for the compound group PERSON, consisting of the item schemas NAME, NUMBER, and SALARY, and the (simple) group schemas SKILL and BIRTH.

Principal group schemas may be of two kinds: repeating and non-repeating. A repeating group schema is one which may have a variable number of instances for each instance of the containing group. By contrast, a non-repeating group schema is one which has just one instance per instance of the containing group. In Figure 2-11, SKILL is a repeating group schema since the number of skills possessed by a person can vary from person to person. BIRTH, on the other hand, is a non-repeating group schema, since every person has exactly one birthdate.

A repeating group consisting of a single item affords an alternative way of achieving a multiple-valued item (see 2.1.3).

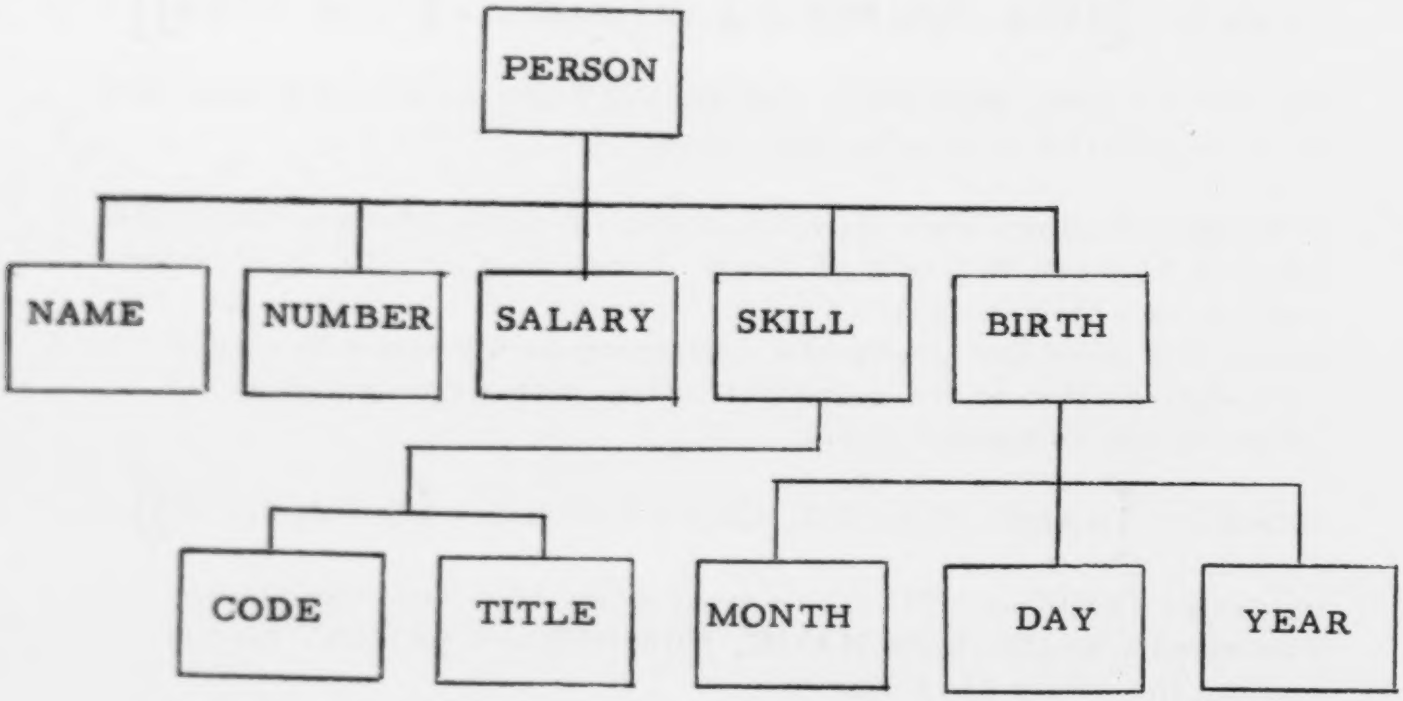

PERSON
NAME
NUMBER
SALARY
SKILL
BIRTH
CODE
TITLE
MONTH
DAY
YEAR

Figure 2-11
A compound group schema

An instance of a compound group schema consists of

- one instance of each principal item schema, plus
- one instance of each non-repeating principal group schema, plus
- zero or more instances of each repeating principal group schema.

Figure 2-12 illustrates an instance of the compound group schema PERSON shown in Figure 2-11.

In a compound group, the set of instances of a repeating principal group schema is referred to as an assembly, and the individual instances in the set are called assembly members. In a repeating group, a set of one or more items whose values serve to uniquely identify the members of an assembly is called the group identifier. Similarly, a set of items whose values serve to order the members of an assembly is called the group sequencer.

In Figure 2-12, the item CODE serves as identifier for the repeating group SKILL, implying that no two members of a SKILL assembly will have the same value of CODE. The SKILL assembly in the PERSON group instance for N. M. Peterson contains three members.

All systems surveyed possess at least one structure type corresponding to the group level, and some have as many as four. Figure 2-13 indicates the terms used by the various systems for group.

2.2.1 Group types

Many systems provide groups of several types, i.e., groups which have different rules of composition or which are used differently in composing other structures. Thus most systems treat repeating and non-repeating groups as different group types. A given system will not normally treat simple and compound groups as separate types, since the simple group is a degenerate case of the compound group.

Figure 2-14 lists the various group types provided by each of the systems, and classifies each as simple or compound, and repeating or non-repeating. Each of the types shown has different rules of composition, or use in higher level structures, as indicated in the sequel.

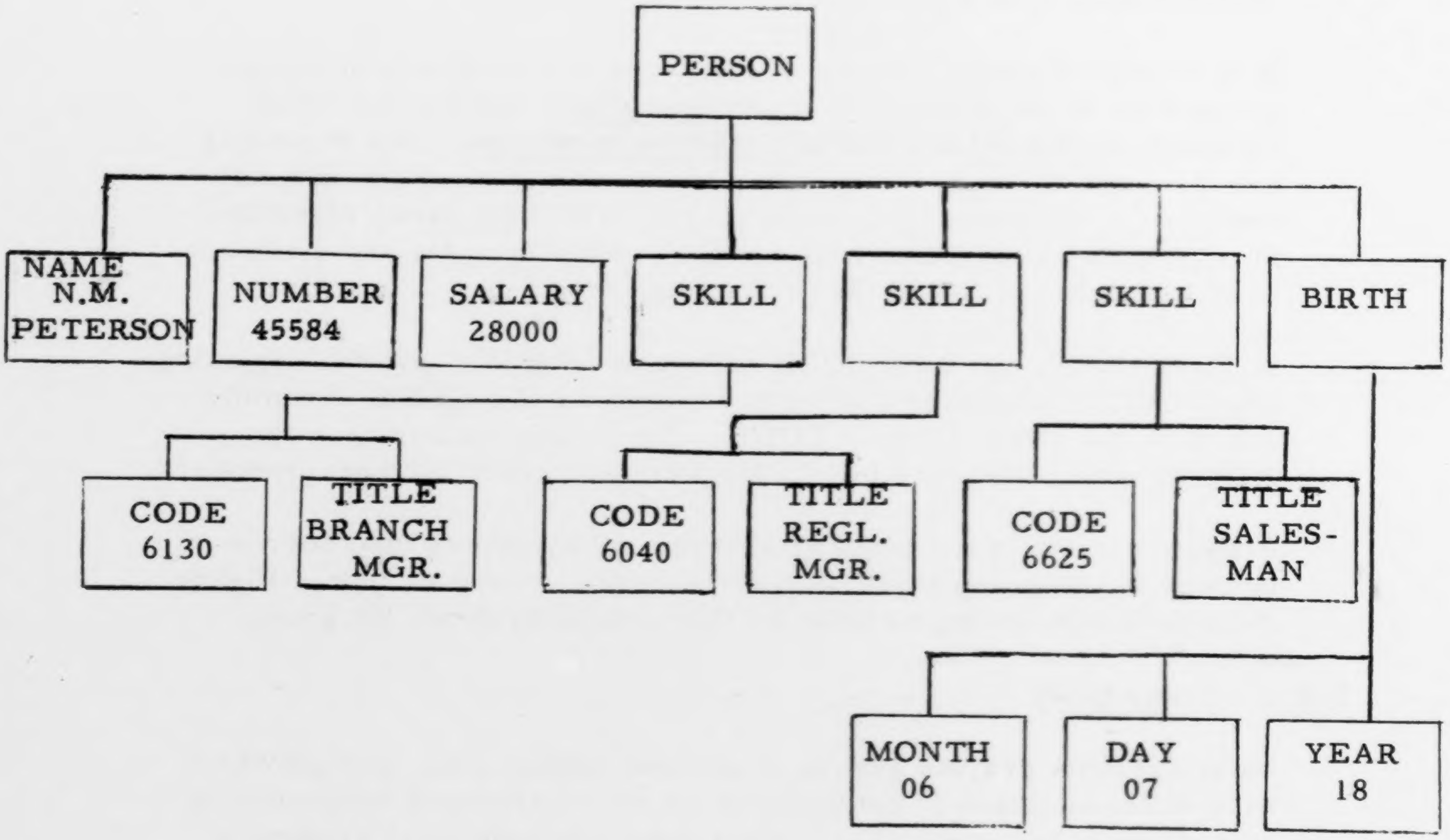

PERSON
NAME N.M. PETERSON
NUMBER 45584
SALARY 28000
SKILL
SKILL
SKILL
BIRTH
CODE 6130
TITLE BRANCH MGR.
CODE 6040
TITLE REGL. MGR.
CODE 6625
TITLE SALES-MAN
MONTH 06
DAY 07
YEAR 18

Figure 2-12
An instance of a compound
group schema

GIS	segment
MARK IV	segment
NIPS/FFS	set, group
TDMS	group
UL/1	group
COBOL	group item
DBTG	record, data aggregate
IDS	record, group item
IMS	segment
SC-1	statement, record

Figure 2-13
System's term for group

	SYSTEM TERM	SIMPLE/ COMPOUND	REPEATING/ NON-REPEATING
GIS	segment	simple	repeating
MARK IV	segment	simple	repeating
NIPS/FFS	fixed set	compound	non-repeating
	periodic set	compound	repeating
	variable set	simple	non-repeating
	group	compound	non-repeating
TDMS	repeating group	compound	repeating
UL/1	repeating group	simple	repeating
	naming group	simple	non-repeating
COBOL	group item	compound	repeating or non-repeating
DBTG	data aggregate	compound	repeating or non-repeating
	record	compound	repeating
IDS	group item	compound	non-repeating
	record	compound	repeating
IMS (DA)	physical segment	simple	repeating
IMS (AP)	logical segment	simple	repeating
SC-1	record	compound	repeating
	statement	compound	non-repeating

Figure 2-14
Group types

2.2.2 Group composition

2.2.2.1 Schema composition

Group schema composition rules will generally depend on group type. The rules include

- limitation on the number and type of principal item schemas in a group schema
- limitation on the number and type of principal group schemas in a compound group schema
- limitation on the depth of nesting in a compound group schema

GIS

A group schema is composed of one or more item schemas.

MARK IV

A group schema is composed of one or more item schemas.

NIPS/FFS

A "group" schema is composed of one or more fixed-length item schemas and "group" schemas.

The "fixed set" and the "periodic set" have the same rules of composition: a group schema is composed of one or more fixed-length item schemas and "group" schemas, and optionally a single "variable length" item schema. The total number of "group" schemas (including nested "groups") cannot exceed 50, and the total number of item and "group" schemas cannot exceed 100.

A "variable set" is composed of one or more item schemas and group schemas.

TDMS

A group schema is composed of one or more item schemas and group schemas.

UL/1

A group schema is composed of one or more item schemas.

COBOL

A group schema is composed of one or more item schemas and group schemas. If a group schema is repeating (see 2.2.3.2), it cannot contain any variable length item schemas or variably repeating group schemas.

DBTG

A "data aggregate" schema and a "record" schema are both composed of one or more item schemas and "data aggregate" schemas. Constituent "data aggregate" schemas may be repeating.

IDS

A "group item" schema and a "record" schema are both composed of one or more item schemas and "group item" schemas.

IMS (DA)

A group schema is seen by the user as a sequence of bytes which may be arbitrarily partitioned into item schemas using the facilities of the application programming language. In addition, IMS itself provides for item definition. Item definitions are required only for items which serve as group identifiers.

Any consecutive set of bytes in the group schema may be defined as an item schema. A given byte may be a component of more than one item schema, or of no item schema.

A maximum of 255 item schemas may be defined in a group schema, and a maximum of 1000 item schemas may be defined in an entry schema.

If the group schema is a dependent in a "logical" group relation schema (see 2.3.1), two sets of item schemas may be defined, one for the group as viewed from the "physical" parent and the other for the group as viewed from its "logical" parent.

IMS (AP)

A group schema in the AP data structure is derived from one or two group schemas in the DA data structure. The first of these schemas is called the "primary source" and the second is called the "secondary source". The secondary source is optional; if present, the primary source must be a dependent in a "logical" group relation, and the secondary source must be the parent of the primary source by either a "physical" or "logical" group relation. The primary and secondary sources need not occur in the same DA entry schema.

A group schema is composed of one set of bytes from the primary source, and one set of bytes from the secondary source. One of the following may be selected from each source:

1) the entire source
2) the identifier within the source

The set from the primary source precedes that from the secondary source.

A group schema has the same item schemas as its source(s).

SC-1

A group schema is composed of one or more item schemas and group schemas.

2.2.2.2 Instance composition

The composition rules for group instances include any limitations imposed by the system on the composition process. An example is the limitation on the number of members in an assembly, or on the ordering of assembly members.

GIS, MARK IV, NIPS/FFS, TDMS, UL/1, COBOL, DBTG, IDS, IMS, SC-1

An instance of a group schema consists of

- one instance of each principal item schema, plus
- one instance of each non-repeating principal group schema, plus
- zero or more instances of each repeating principal group schema.

2.2.3 Group attributes

2.2.3.1 Names

Provision is normally made for assigning names or other identifiers to groups. A group name is used to refer collectively to the items in the group, or to qualify an otherwise ambiguous reference to an item in the group.

In most systems, group names are alphabetic. A few systems require group numbers to be assigned, either in place of or in addition to alphabetic names. See Figure 2-15.

Only COBOL and IDS make explicit provision for group synonyms (through the RENAMES facility, which is restricted to non-repeating groups). In other systems, the effect of synonyms can often be achieved through group re-definition.

2.2.3.2 Other schema attributes

Other possible schema attributes of a group include

- item sets which serve in a special capacity for the group, such as group identifier or group sequencer
- access locks for controlling access to the contained item values
- codes which identify the programs making use of the group

Figure 2-16 indicates the facilities provided in the surveyed systems for associating sequencers with repeating groups, and whether or not the sequencer may or must also be the group identifier (i.e., whether the sequencer values in an assembly must be unique). None of the systems provide for independent sequencers and identifiers.

Other schema attributes not shown in the figure are as follows:

GIS

A group schema must contain one "count" item schema for each dependent group schema. The value of a count item is the number of instances of the dependent group schema existing at any moment. It must be declared by the user, but is maintained by the system.

	GROUP TYPE	NAME TYPE		UNIQUE WITHIN	SYNONYMS
		ALPHA	NUMBER		
GIS	all	1 to 8 char.	none	file	no
MARK IV	all	none	1-99	file	no
NIPS/FFS	fixed set	"X"	0	file	no
	periodic set	none	1-255	file	no
	variable set	1 to 7 char.	(1)	file	no
	group	1 to 7 char.	none	file	no
TDMS	all	1 to 255 char.	1-32755	file	no
UL/1	repeating gr.	1 to 8 char.	none	file	no
	naming gr.	1 to 8 char.	none	file	no
COBOL	all	1 to 30 char.	none	group	yes[2]
DBTG	data aggregate	1 to 30 char.	none	group	no
	record	1 to 30 char.	none	file	no
IDS	group item	1 to 30 char.	none	group	yes[2]
	record	1 to 30 char.	1-999	file	no
IMS	all	1 to 8 char.	none	file	no
SC-1	all	2 to 31 char.	none	group	no

[1] Consecutive numbers following the highest "periodic set" number are assigned by the system.

[2] For non-repeating groups only.

Figure 2-15
Group names

	REQ'D/OPT.	NO. OF ITEMS	ASC. /DESC.	USE AS IDENTIFIER
GIS	required[1]	1 to 7	A or D separately	optional
MARK IV	required	1 to 9	A	required
NIPS/FFS	required[2]	1 or more[3]	A	required
TDMS	no sequencer or identifier attribute provided			
UL/1	no sequencer or identifier attribute provided			
COBOL	optional	1 or more	A or D	no
DBTG	no sequencer or identifier attribute provided[4]			
IDS	no sequencer or identifier attribute provided[4]			
IMS	optional[5]	1	A	required
SC-1	optional[6]	1 or more	A or D separately	no

[1] Sequencer is optional in groups at lowest level in entry.

[2] If user does not specify a sequencer, system supplies a 4-character serial number.

[3] Total length of group sequencer and entry sequence must not exceed 244 characters.

[4] Sequencer and identifier attributes may be associated with the group relation(s) in which the "record" is a dependent (see 2.3.3.2).

[5] Sequencer is required in entry-defining group of indexed files.

[6] Sequencer is for documentation purposes only.

Figure 2-16
Repeating group sequencer attribute

MARK IV

A group schema must contain a "count" item schema for each dependent repeating group schema, as in GIS.

NIPS/FFS

A "group" may be assigned one of the types "numeric", "alphanumeric", or "coordinate" as long as it doesn't conflict with the types of the constituent items.

TDMS

None.

UL/1

None.

COBOL

A group schema is repeating if it has the OCCURS attribute. The OCCURS attribute specifies the number of instances of the schema that can occur within a given parent group. The schema is fixed repeating if this number is a constant, and variably repeating if the number is a variable.

An "index" may be associated with a repeating group schema for purposes of accessing the instances of the schema.

DBTG

A "data aggregate" schema may have the OCCURS attribute as in COBOL.

"Record" schema attributes include:

(1) a "location mode" which determines the method used to access the "record" ("direct" through a system identifier, "via" a relation with another "record", or through a system identifier "calculated" from a randomizing algorithm).

(2) for "location mode" = "via", the name of the group relation used to access the "record" (see 2.3.2.1).

(3) the "areas" within which the instances of the "record" schema occur (see 2.7).
(4) a procedure to be executed whenever the "record" is subject to one or more of a fixed set of operations, such as "insert", "remove", etc.
(5) access locks for one or more of a fixed set of operations on a "record".

IDS

The "record" schema has the following attributes:

(1) an "access type" which is like the "location mode" in DBTG, except that "direct" and "via" are known as "primary" and "secondary", respectively.
(2) for "access type" = "secondary", the name of the group relation used to access the "record".
(3) access locks.

IMS (DA)

The group schema requires a length attribute, since the declaration of items is optional. Other attributes include insert, replace and delete rules for groups which participate in "logical" group relations, and ordering rules for groups with non-unique sequencer or no sequencer.

A group which is a dependent in a "logical" group relation may have two sets of schema attributes, one for the "physical" group relation and one for the "logical" group relation in which it participates.

IMS (AP)

The sequencer/identifier from the group's primary source, or the concatenated sequencer/identifiers from its primary and secondary source, becomes the group's sequencer/identifier. Other attributes include length; insert, replace and delete rules; and ordering rules. The latter two attributes are inherited from the group's primary source.

In the context of a given program, a group may have an access type of "read-write" or "read only".

SC-1

A repeating group schema assembly, called a "subfile", may be assigned a name of the same form as the group name. The assembly name is used for referring to all members of the assembly collectively.

2.2.3.3 Non-schema attributes

Non-schema attributes of groups include

- date and time of group insertion
- count of references to the group

GIS, MARK IV, NIPS/FFS, TDMS, UL/1, COBOL, IMS

None.

DBTG

A system identifier, a "database key", is associated with each "record". The identifier may be used to access "records" with "location mode" equal to "direct".

IDS

A system identifier, a "record reference code", is associated with each "record". It may be used to access records with "access type" equal to "primary".

SC-1

A system identifier, the "item position code", is associated with each group, and may be used to access groups.

2.3 Group relations

A group relation is a relation or mapping between two sets of groups. The groups in the first set are called parent groups, while those in the second are called dependent groups. A group relation may have a name and other attributes of its own.

The group relation provides a way of relating groups, hence of relating the application entities to which these groups correspond. For example, given a set of PERSON groups:

{PERSON(G. J. GUTTMAN), PERSON(R. J. WALTERS)}

and a set of SKILL groups:

{SKILL(1110), SKILL(1120), SKILL(1130), SKILL(1135)},

the following group relation can be established to relate people to the skills which they possess:

{
<PERSON(G. J. GUTTMAN), SKILL(1110)>,
<PERSON(G. J. GUTTMAN), SKILL(1135)>,
<PERSON(R. J. WALTERS), SKILL(1110)>,
<PERSON(R. J. WALTERS), SKILL(1120)>,
}

In this relation, the PERSON groups are the parent groups, and the SKILL groups are the dependent groups. Note that all groups do not necessarily have to participate in the relation; in particular, SKILL (1130) is not associated with any PERSON.

A group relation also provides a way of establishing a hierarchic relation between two sets of items. In particular, the items in a dependent group stand in a subordinate relation to the items in an associated parent group. In a hierarchic group relation, each dependent group must be associated with one parent group; the dependent group cannot exist meaningfully by itself. The following is a hierarchic group relation between a group representing a person and a set of groups representing the degrees he has earned:

{
<PERSON(G. J. GUTTMAN), DEGREE(AB, 1955, PRINCETON)>
<PERSON(G. J. GUTTMAN), DEGREE(MS, 1957, MIT)>
}

The hierarchic relation facility provided by a group relation is equivalent to that provided by a compound group (see 2.2), except for the following:

1) In a compound group, a group may be subordinated to a single set of items only (the principal items) whereas in a group relation a group may be subordinated to many sets of items (parent groups).

2) In a compound group, the items to which a group is subordinated (i.e., the principal items) do not have a collective name (recall that the compound group name refers to all contained items). In a group relation, on the other hand, each set of items to which a group is subordinated may have a name of its own, namely, the parent group name.

The relation expressed in a group relation is known as a binary relation, since it expresses a relation among the objects of two sets. The more general case of the n-ary relation, i.e., the relating of objects from n sets, is not considered further here because few (if any) systems make explicit provisions for such relations.

A group relation schema consists of one or more parent group schemas and one or more dependent group schemas. Such a schema can be portrayed as a directed graph containing a node for each parent and dependent group schema, and a single forked arc with multiple tails and heads - a tail leaving each parent group schema and a head entering each dependent group schema. The arc can be labelled with the relation name, if it has one.

Figure 2-17 illustrates a common class of group relation schema in which there is only a single parent group schema and a single dependent group schema.

Figure 2-18 illustrates a group relation schema in which the same group schema serves as both parent and dependent. Such schemas are used to represent relations among a homogeneous set of entities.

Figure 2-19 illustrates a group relation schema having multiple dependent group schemas. This class of schema is useful when it is necessary to treat the instances of several different group schemas either as members of a single set or as members of several different sets, depending on the context. Thus, the schema of Figure 2-19 will permit the useful skills and useless skills of a person to be processed independently in one context (e.g., list all useful skills and all useless skills) and jointly in another (e.g., list all skills in order of decreasing proficiency).

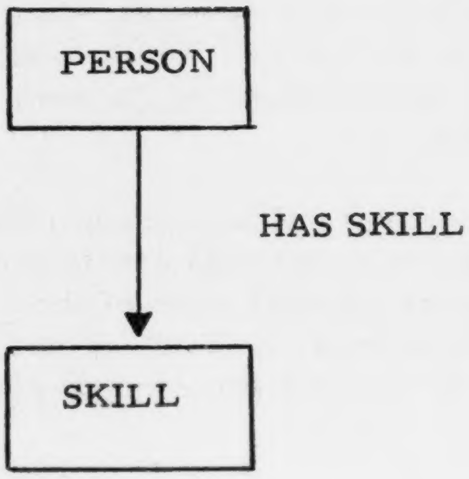

PERSON
HAS SKILL
SKILL

Figure 2-17
A group relation schema

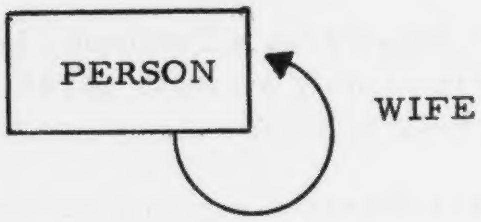

PERSON
WIFE

Figure 2-18
A group relation schema
with one group schema

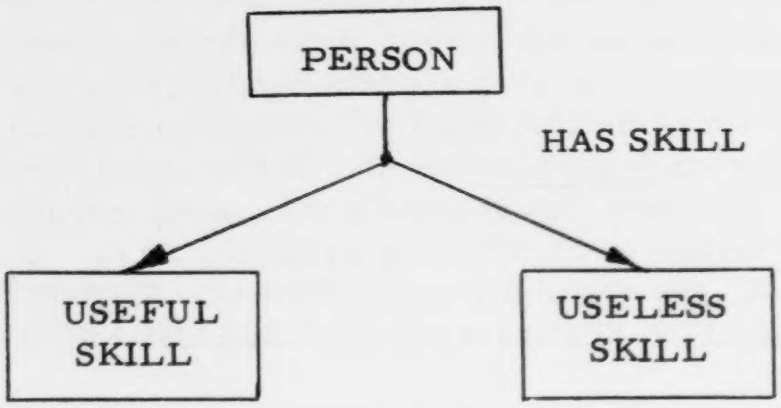

PERSON
HAS SKILL
USEFUL SKILL
USELESS SKILL

Figure 2-19
A group relation schema with
multiple dependent group schemas

An instance of a group relation schema consists of one or more instances of each parent and dependent group schema, with each parent group instance being optionally paired with one or more dependent group instances. If the group relation is *non-hierarchic*, each dependent group instance may be optionally paired with one or more parent group instances. If the group relation is *hierarchic*, each dependent group *must* be paired with one parent. Figures 2-20, 2-21, and 2-22 illustrate instances of the group relation schemas of Figures 2-17, 2-18, and 2-19, respectively.

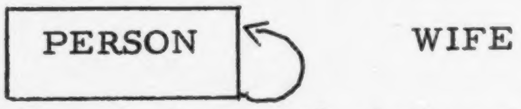
In analogy with compound groups, a dependent group schema in a group relation schema may be *repeating* or *non-repeating*, depending on whether there may be a variable number or just one instance of the group schema paired with a given parent group. Also, the dependent groups paired with a given parent group are called an *assembly*, and may be identified or ordered by the values of one or more *identifier* or *sequencer* items. Unlike an assembly in a compound group, however, an assembly in a group relation may be composed of instances of different group schemas. For example, in Figure 2-22, the assembly associated with PERSON(G.J. GUTTMAN) consists of USEFUL SKILL(1110) and USELESS SKILL(X173).

In a given system, the group relation may be an explicitly identified structure type, or it may be an implied part of a larger structure type. In either case, the structures possible in the system can be described by stating the restrictions it places on the composition of group relation schemas and group relation instances. Examples of group relation schema restrictions are

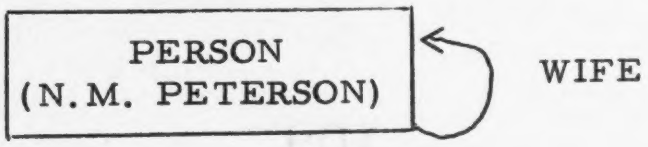
- Single parent group schema and single dependent group schema only
- Parent group schema(s) must be distinct from dependent group schema(s) (i.e., no "loops").

Examples of group relation instance restrictions are

- Parent group instance and dependent group instance must be distinct. For example, the schema



might be allowed, but not the instance



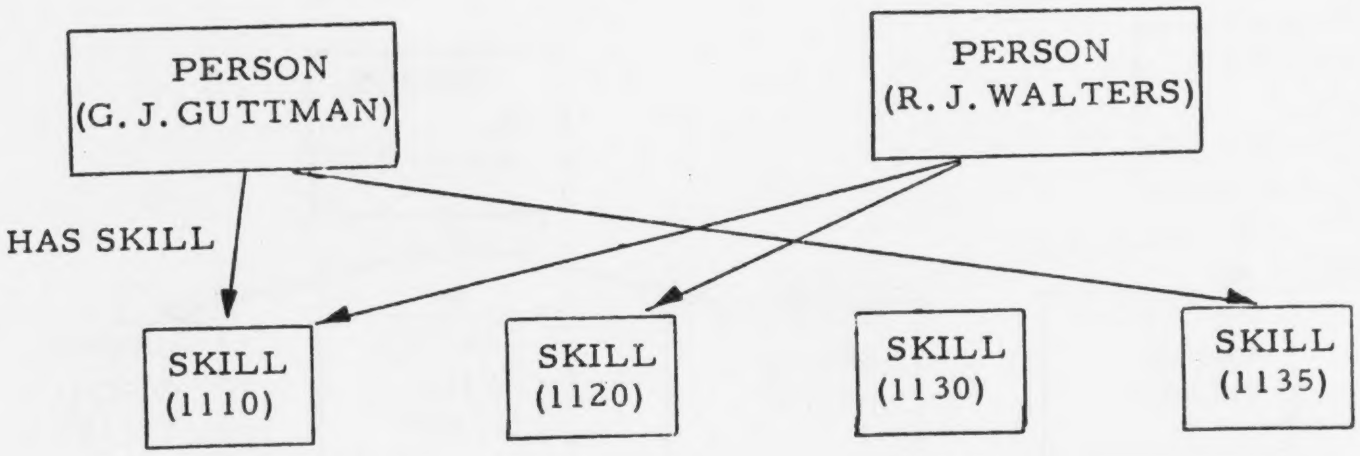

PERSON
(G. J. GUTTMAN)
PERSON
(R. J. WALTERS)
HAS SKILL
SKILL
(1110)
SKILL
(1120)
SKILL
(1130)
SKILL
(1135)

Figure 2-20
An instance of the group relation schema
of Figure 2-17

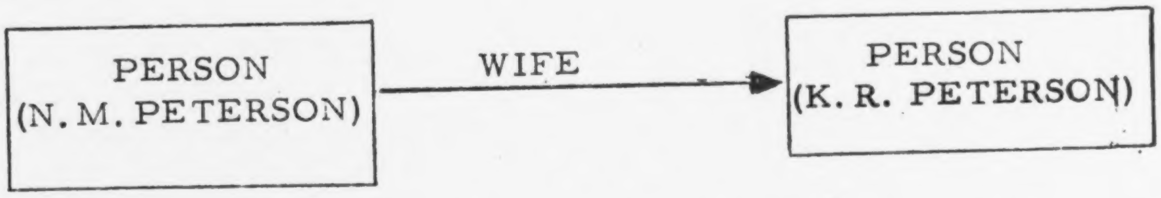

PERSON
(N. M. PETERSON)
WIFE
PERSON
(K. R. PETERSON)

Figure 2-21
An instance of the group relation schema
of Figure 2-18

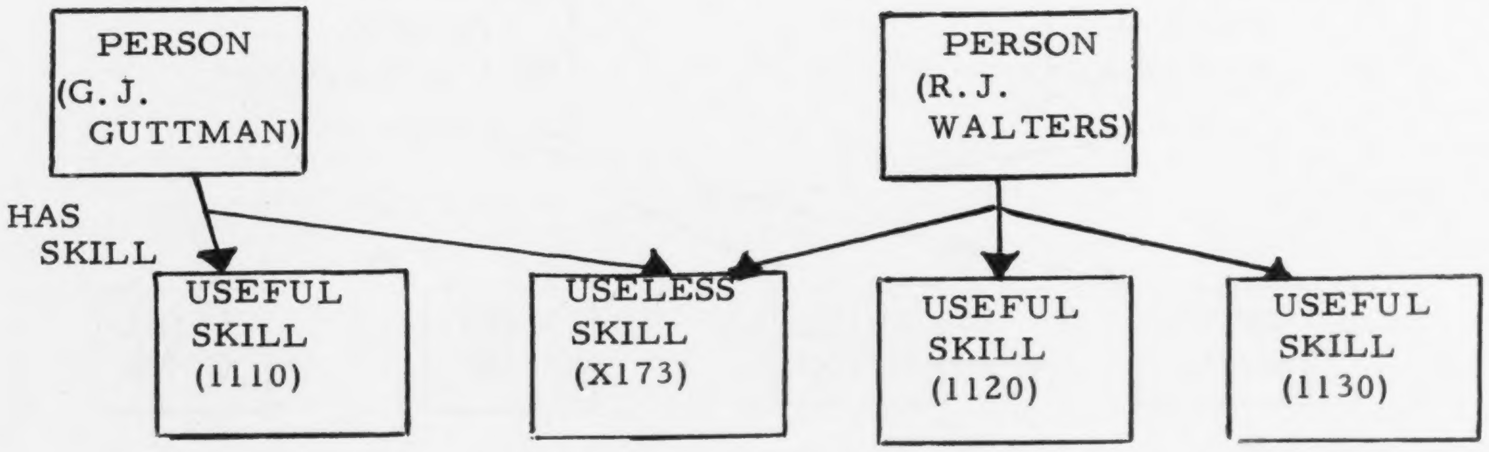
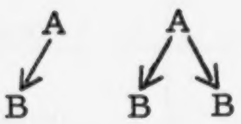

PERSON
(G. J. GUTTMAN)
PERSON
(R. J. WALTERS)
HAS SKILL
USEFUL SKILL (1110)
USELESS SKILL (X173)
USEFUL SKILL (1120)
USEFUL SKILL (1130)

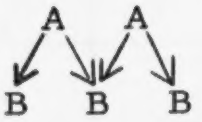
Figure 2-22
An instance of the group relation schema
of Figure 2-19

- A dependent group instance has no more than one associated parent group instance. Thus:

Allowed



Not Allowed



Of the systems surveyed, only DBTG, IDS and IMS make explicit provision for group relations. In three other systems - GIS, MARK IV, UL/1 - the group relation structure is implied in the systems' facility for tree structures. In the remaining systems, the group relation structure is not provided. Table 2-23 illustrates the situation.

2.3.1 Group relation types

A system may provide more than one type of group relation, i.e., group relations which have different rules of composition or are used differently in composing other structures.

GIS, MARK IV, UL/1, IMS(AP)

Each of these systems provides implicitly a single type of group relation.

DBTG

DBTG provides a single group relation type, the "set".

IDS

IDS provides two group relation types, referred to as the "chain" and the "CALC chain".

IMS(DA)

Two group relation types are provided, "physical" and "logical".

NIPS/FFS, TDMS, COBOL, SC-1

Not applicable.

	EXPLICIT/IMPLICIT STRUCTURE	SYSTEM TERM
GIS	implicit	none
MARK IV	implicit	none
NIPS/FFS	structure not provided	none
TDMS	structure not provided	none
UL/1	implicit	none
COBOL	structure not provided	none
DBTG	explicit	set
IDS	explicit	chain
IMS(DA)	explicit	relationship
IMS(AP)	implicit	none
SC-1	structure not provided	none

Figure 2-23
System's term for group relation

2.3.2 Group relation composition

2.3.2.1 Schema composition

Composition rules for group relation schemas include

- limitation on the number of parent group schemas and dependent group schemas
- requirement that the parent group shcemas(s) be distinct from dependent group schemas(s).

GIS, MARK IV, IMS

A group relation schema consists of a single parent group schema and a single dependent group schema. The parent and dependent group schemas must be distinct.

UL/1

A group relation schema consists of a single "repeating group" schema as parent and a single "repeating group" schema as dependent. The parent and dependent group schemas must be distinct.

DBTG

A group relation schema consists of a single parent group schema and one or more dependent group schemas. The parent and dependent group schemas must be of the type "record" and must be distinct.

A parent group is called an "owner record" and a dependent group is called a "member record."

A special group relation schema is provided which has no parent group schema. Its parent is, in effect, the system.

IDS

A "chain" schema has the same rules of composition as in DBTG. A parent group is called a "master record" and a dependent record is called a "detail record".

A "CALC chain" schema is a "chain" schema with no parent group schema. Its parent is, in effect, the file schema.

NIPS/FFS, TDMS, COBOL, SC-1

Not applicable.

2.3.2.2 Instance composition

This includes the rules for composing group relations from groups, together with any restrictions on this process such as the requirement that a group have no more than one parent.

GIS, MARK IV, UL/1, IMS

An instance of a group relation schema consists of one instance of the parent group schema and zero or more instances of the dependent group schema, with each dependent group paired with the parent group. The group relation is thus hierarchic.

DBTG, IDS

An instance of a group relation schema consists of one instance of the parent group schema and zero or more instances of the dependent group schema, with each dependent group optionally paired with the parent group. The relation is thus non-hierarchic.

NIPS/FFS, TDMS, COBOL, SC-1

Not applicable.

2.3.3 Group relation attributes

2.3.3.1 Names

This is the facility for assigning names or other identifiers to group relations.

GIS, MARK IV, UL/1, IMS

No naming facility is provided.

DBTG, IDS

Group relations must be assigned names of 1 to 30 characters, unique within the file. In IDS, group relations of the type "CALC chain" have the name "CALC".

NIPS/FFS, TDMS, COBOL, SC-1

Not applicable.

2.3.3.2 Other schema attributes

Other schema attributes for a group relation include

- item sets which serve in a special capacity, such as dependent group identifier.
- access locks for controlling access to parent or dependent groups
- placement criteria for new assembly members, e.g., first, last, or in order by sequencer item values.

GIS, MARK IV, UL-1, IMS

None.

DBTG, IDS

A group relation schema may have an ordering attribute, which determines the ordering of dependent groups in instances of the relation. Ordering may be based on order of group insertion, or on any combination of dependent group item values, dependent group name, or dependent group sequencer.

A group relation schema may also have a separate identifier attribute, consisting of a set of items whose values uniquely identify dependent groups in instances of the relation. In IDS, this facility is limited to the "CALC chain" relation.

In DBTG, each dependent group schema in a relation schema may have an "automatic/manual" attribute. An automatic group is automatically included in a dependent assembly whenever the group is STORED. A manual group will be so included only through the INSERT verb. A relation may additionally have a "mandatory/optional" attribute. A mandatory group can be deleted only by the DELETE verb. An optional group can be deleted either by a DELETE SELECTIVE or REMOVE verb. IDS has no comparable facility.

The parent group schema in a relation schema may have an identifier attribute, consisting of a set of item names whose values are used to identify instances of the relation.

In DBTG, a relation schema may have access locks separate from the locks on the constituent groups. No such facility is provided in IDS.

Note that each of the foregoing attributes is associated with the relation, rather than with the group. This enables a given group which is the dependent group in two or more relations to have different sets of attributes in each relation.

NIPS/FFS, TDMS, COBOL, SC-1

Not applicable.

2.3.3.3 Non-schema attributes

An example of a non-schema attribute for group relations is the count of the members in an assembly.

No non-schema attributes are associated with group relations in the systems surveyed.

2.4 Entries

An entry is a set of groups and group relations in which one and only one group, the entry-defining group, is not contained in or subordinate to any other group. The entry corresponds roughly to "record", a term which is not used here because of possible confusion with the storage structure of the same name.

The entry is used to represent the major entities of an application. For a given class of entities (e.g., the employees of a firm), the items or principal items in the entry-defining group typically correspond to fixed entity attributes, i.e., attributes common to all entities in the given class; while the items in the contained or subordinate group correspond to the variable entity attributes, i.e., attributes not necessarily shared by all entities in the class or which may have multiple values.

The entry-defining group of an entry may not be the dependent group in a hierarchic group relation. It may, however, be the dependent group in any non-hierarchic group relation for purposes of relating entries in the same or different files. Facilities for accomplishing inter-entry relations are covered in the sequel (see 2.5 and 2.6).

Three major types of entry can be defined: the group entry, the tree entry, and the plex entry.

A group entry consists of a single compound group. The hierarchic relating of items is achieved through group nesting, as described in Section 2.2.

A group entry schema coincides with the schema of the entry-defining group. An instance of a group entry schema is just an instance of the entry-defining group schema.

Figure 2-24 illustrates a group entry schema.

A *tree entry* is a set of hierarchic group relations arranged as a *tree,* i.e., arranged such that each group has at most one parent, and one and only one group has no parent. The group without a parent is the entry-defining group. It is also referred to as the *root group*.

In a tree entry, the hierarchic relating of items is achieved primarily through group relations, as described previously (see 2.3). The tree entry may contain compound groups, but the principal groups in any such compound groups are typically restricted to being non-repeating.

As a special case, the tree entry may consist of a single group having no dependent groups. For this case, the tree entry is identical to the group entry.

A tree entry schema consists of a single group schema, or of one or more hierarchic group relation schemas arranged as a tree. An instance of a tree entry schema is a tree consisting of one instance of the entry-defining group schema, and zero or more instances of each dependent group schema for each instance of its parent.

Figure 2-25 illustrates a tree entry schema containing the same data as the group entry schema of Figure 2-24. The group relations in a tree entry are typically unnamed, since only a single kind of relation between groups is being represented, namely, the hierarchic relation.

A *plex entry* is a set of group relations in which every group except one, the entry-defining group, is the dependent in a hierarchic group relation. Additionally, all groups in a plex entry may participate in any number of non-hierarchic group relations.

The plex entry is a generalization of a tree entry, which results from allowing non-hierarchic relations to be established between any pair of groups. As in the tree entry, the groups in a plex entry tend to be simple groups, but are not necessarily so. The plex entry includes the special case of a single root group.

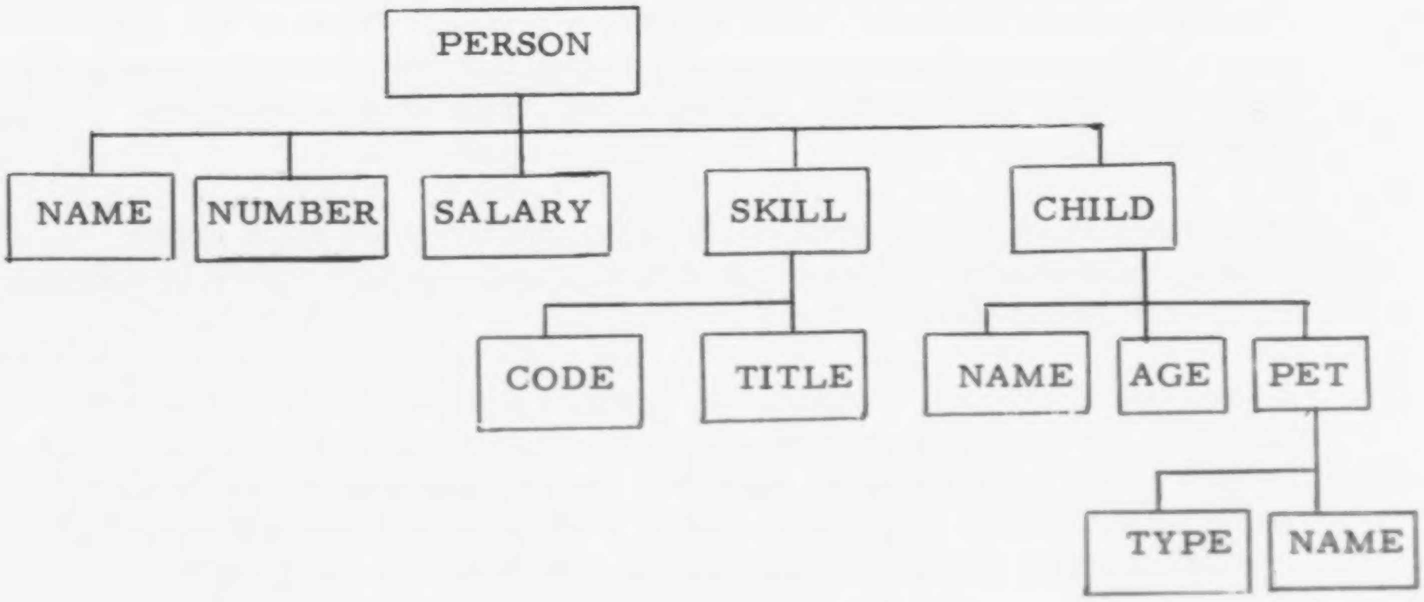

PERSON
NAME
NUMBER
SALARY
SKILL
CHILD
CODE
TITLE
NAME
AGE
PET
TYPE
NAME

Figure 2-24
A group entry schema

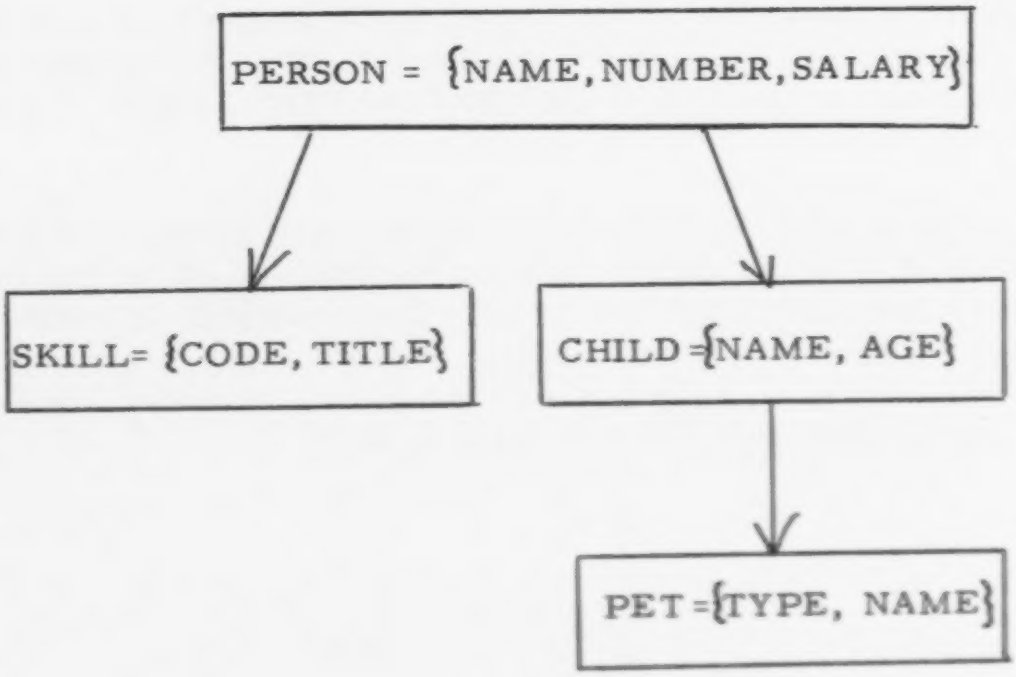
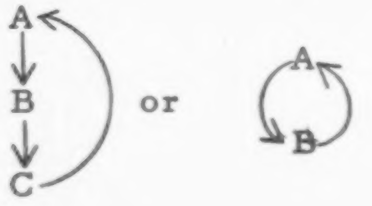

PERSON = {NAME, NUMBER, SALARY}
SKILL= {CODE, TITLE}
CHILD ={NAME, AGE}
PET ={TYPE, NAME}

Figure 2-25
A tree entry schema

A plex entry schema consists of a single group schema, or of one or more group relation schemas in which one and only one group relation schemas in which one and only one group schema, the entry-defining group schema, is not the dependent in a hierarchic group relation schema. An instance of a plex entry schema consists of one instance of the single entry-defining group schema, or of zero or more instances of each constituent group relation schema subject to the restriction that there be only one instance of the entry-defining group schema.

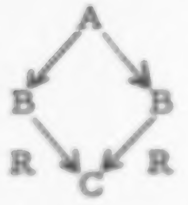
Figure 2-26 illustrates a plex entry schema obtained from the tree entry schema of Figure 2-25 by adding the non-hierarchic relation FEEDS between the groups PERSON and PET. Figure 2-27 illustrates an instance of this schema.

In addition to restrictions on the formation of group relation schemas, a system may place restrictions on the manner of incorporating such schemas into a plex entry schema. For example, in addition to prohibiting "loops" (i.e., relations in which one group schema serves as both parent and dependent, a system may prohibit "cycles", i.e., paths that close on themseleves as in

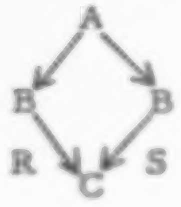


Similarly, the system may place additional restrictions on the formation of plex entry instances. For example, the system may prohibit a group from having multiple parents by the same relation. Thus, if R and S are different relations,

Not Allowed



Allowed



In any of the foregoing types of entry, the entry-defining group schema is generally considered to be a repeating group schema, in the sense that it can have multiple instances for each instance of the containing structure (the file). The set of instances of an entry

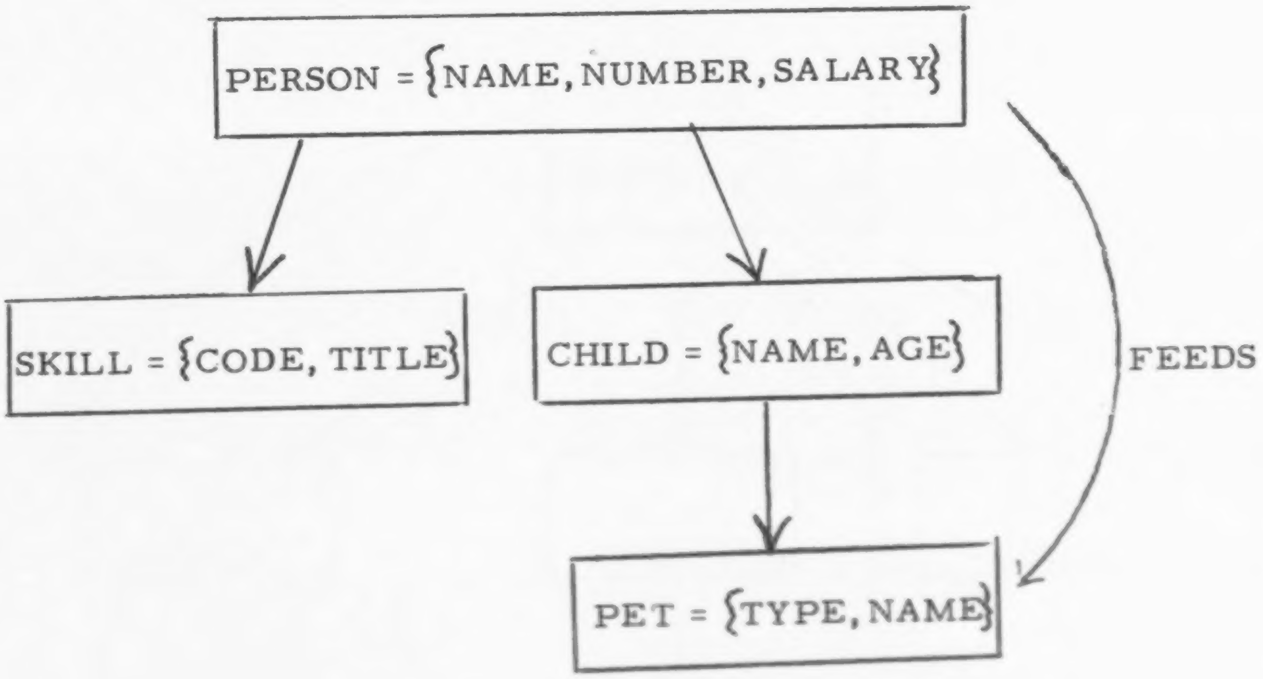

PERSON = {NAME, NUMBER, SALARY}
SKILL = {CODE, TITLE}
CHILD = {NAME, AGE}
FEEDS
PET = {TYPE, NAME}

Figure 2-26
A plex entry schema

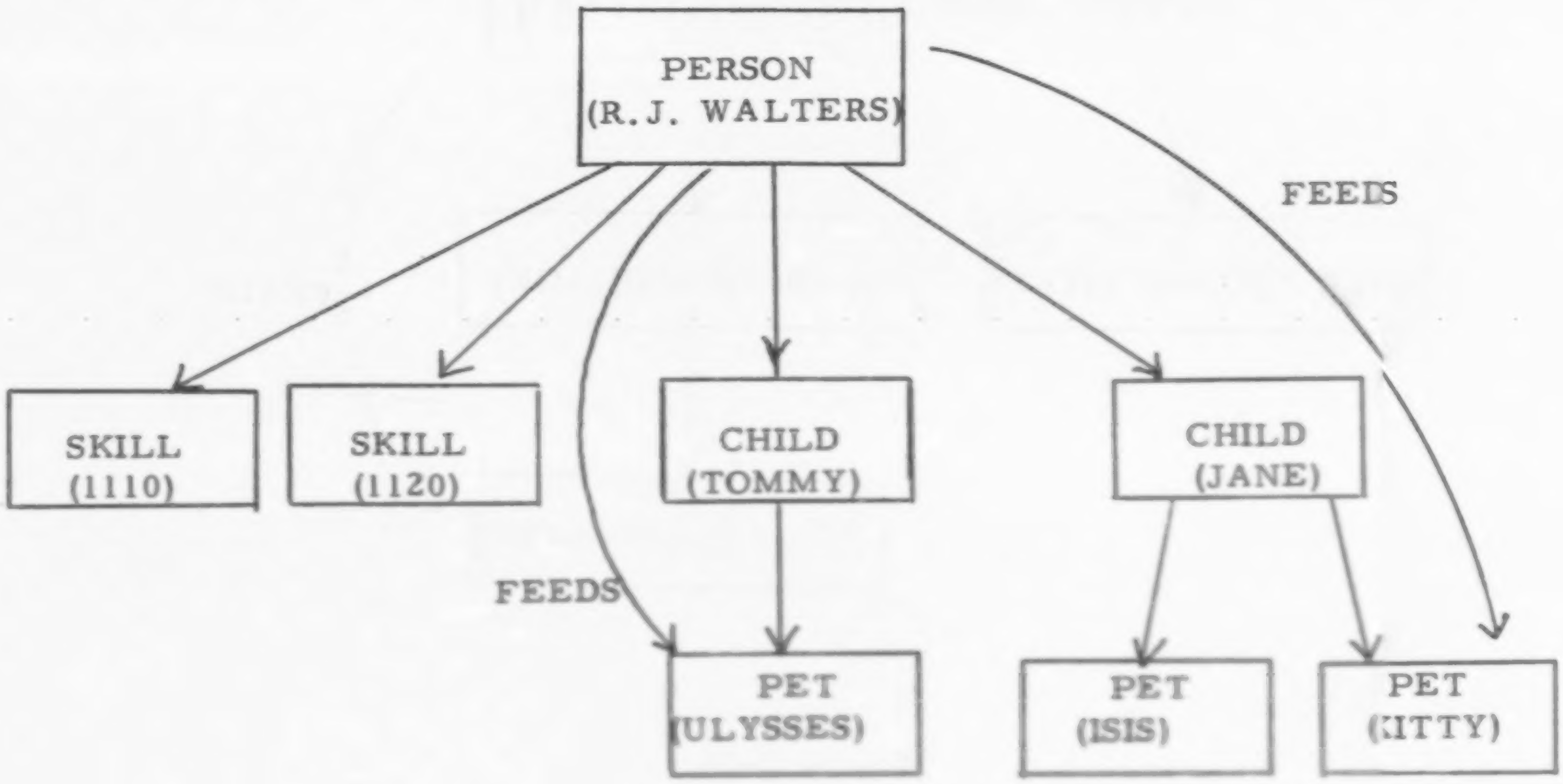

PERSON
(R.J. WALTERS)
FEEDS
SKILL
(1110)
SKILL
(1120)
CHILD
(TOMMY)
CHILD
(JANE)
FEEDS
PET
(ULYSSES)
PET
(ISIS)
PET
(KITTY)

Figure 2-27
An instance of the plex entry schema
of Figure 2-26

schema in a file are referred to as an assembly, and one or more items in the entry-defining group may be used to identify or order the members of the assembly.

All systems specifically provide the entry structure type. The systems' terms for entry are shown in Figure 2-28.

2.4.1 Entry types

Two entries are of different type if they have different rules of composition or are used differently in composing other structures. Entry type does not correspond to "record type". The latter term is often used to denote an entry schema.

All systems surveyed provide a single type of entry only, as indicated in Figure 2-28. In a NIPS/FFS entry, the "fixed set" can be interpreted either as the entry-defining group in a tree entry or as a set of principal items in a group entry. The latter interpretation has been arbitrarily chosen.

2.4.2 Entry composition

2.4.2.1 Schema composition

Composition rules for entry schemas include

- limitation on number and type of group and group relation schemas comprising the entry schema
- limitation on the number of levels in the entry

GIS

An entry schema is a set of group relation schemas structured as a tree. Multiple group schemas may appear on a single level only at the lowest level of the tree.

The number of levels is limited to 16. The number of group schemas is limited to 255.

MARK IV

An entry schema is a set of group relation schemas structured as a tree.

The number of levels is limited to 9. The number of group schemas is limited to 99.

	SYSTEM TERM	ENTRY TYPE
GIS	record	tree
MARK IV	record	tree
NIPS/FFS	record	group
TDMS	entry	group
UL/1	record	special (see text)
COBOL	record	group
DBTG	record	group
IDS	record	group
IMS(DA)	record	plex
IMS(AP)	record	tree
SC-1	record	group

Figure 2-28
System's term for entry

NIPS/FFS

An entry schema consists of one "fixed set" schema, and zero to 255 "periodic set" and "variable set" schemas.

TDMS

An entry schema has the same composition as a group schema.

The number of levels is limited to 16. The number of group schemas in an entry schema must not exceed 255.

UL/1

An entry schema consists of zero or more item schemas and zero to 15 sets of group relation schemas, each structured as a tree.

In each tree, the number of levels is limited to 15. The number of group schemas in all trees must not exceed 255.

COBOL

An entry schema has the same composition as a group schema.

The number of levels is limited to 49.

DBTG, IDS

An entry schema has the same composition as a "record" schema (see 2.2.2.1).

IMS(DA)

An entry schema is a set of group relation schemas with the following conditions:

1) Every group schema except one must be the dependent in exactly one "physical" group relation schema. The exception is the entry root; it has no parent.

2) Every group schema except the root may also be the dependent in exactly one "logical" group relation schema.

The number of levels is limited to 15. The number of group schemas and the number of "logical" group relation schemas are each limited to 255.

As a special case, an entry schema may consist of a single root group schema.

IMS(AP)

An entry schema is a set of group relation schemas meeting the following conditions:

1) Every group schema except one must be the dependent in exactly one group relation schema. The exception is the root; it has no parent.

2) The source of the root group schema must be a root group schema in the "physical" entry.

3) If group schema Y is a dependent of group schema X, then the primary source of Y must be a dependent of the primary or secondary source of X.

The number of levels is limited to 15. The number of group schemas cannot exceed 255.

As a special case, an entry schema may consist of a single root group schema.

SC-1 (DA)

An entry schema has the same composition as a group schema.

The number of levels cannot exceed 64. The number of principal item schemas and principal group schemas together cannot exceed 255.

SC-1 (AP)

An entry schema has the same composition as a group schema. An entry schema may correspond to any "subtree" of the DA entry schema which has a repeating group schema for its root.

2.4.2.2 Instance composition

Composition rules for entry instances include

- limitation on number of group or group relation instances
- limitation on size of assemblies and number of relations in which a dependent group may participate.

GIS, MARK IV

An instance of an entry schema consists of one or more instances of each constituent group relation schema, subject to the following restrictions:

1) There is only one instance of the root group schema.
2) Every group except the root must be the dependent in exactly one instance of a group relation schema.

NIPS/FFS

An instance of an entry schema consists of one instance of the "fixed set" schema, zero or more instances of each "periodic set", and zero or one instance of each "variable set" schema.

TDMS, COBOL, DBTG, IDS, SC-1

An instance of an entry schema consists of one instance of the entry-defining group schema.

UL/1

An instance of an entry schema consists of one instance of each principal item schema, and zero or more instances of each tree of group relation schemas.

IMS(DA)

An instance of an entry schema consists of one or more instances of each constituent group relation schema, subject to the following restrictions:

1) There is only one instance of the root group schema.
2) Every group except the root must be the dependent in exactly one instance of a group relation schema.
3) Every group whose schema is the dependent in a "logical" group relation schema must be paired with a single instance of the logical parent schema.

IMS(AP)

An instance of an entry schema consists of one or more instances of each constituent group relation schema, subject to the following restriction:

1) There is only one instance of the root group schema.
2) Every group except the root must be the root must be the dependent in exactly one instance of a group relation schema.
3) If group Y is a dependent of group X, then the primary source of Y must be a dependent of the primary or secondary source of X.

2.4.3 Entry attributes

2.4.3.1 Names

Names may be assigned to entries so that instances of two or more entry schemas can be distinguished when they occur together in the same file. Entry name corresponds roughly to "record type code".

GIS, MARK IV, NIPS/FFS, TDMS, UL/1, IMS

No facility is provided for associating a name with an entry schema.

COBOL

An entry schema must be assigned a name of 1 to 30 characters, unique in the file.

DBTG, IDS

The name of the entry-defining group is the entry name.

SC-1

An entry schema must be assigned a name of 2 to 31 characters.

2.4.3.2 Other schema attributes

Other schema attributes for an entry include

- item sets in the entry-defining group which serve in a special capacity, such as entry identifier
- access locks for controlling access to entry schema instances
- codes which identify the programs making use of the entry.

GIS, MARK IV, IMS, SC-1

In these systems, any identifier or sequencer attributes of the entry-defining group are taken as the identifier or sequencer attributes of the entry (see 2.2.3.2).

NIPS/FFS

An identifier/sequencer comprised of one or more principal items in the "fixed set" is required. The items must be of type "alphameric" and their combined length must not exceed 244 characters.

TDMS

No other entry schema attributes are provided.

UL/1

An identifier/sequencer of 1 to 4 single-valued principal items in the entry schema is required .

COBOL

No other entry schema attributes are provided.

DBTG, IDS

Entry schema attributes include those defined earlier for "records" (see 2.2.3.2).

2.4.3.3 Non-schema attributes

Included in non-schema entry attributes are

- date and time entry was placed in the file
- count of references to the entry.

GIS, MARK IV, NIPS/FFS, TDMS, COBOL, IMS

None.

UL/1

A date stamp is provided for entry instances.

DBTG, IDS, SC-1

The system identifier of the entry-defining group may be used as an entry instance identifier.

2.5 Files

A file is a set of entries. A file thus corresponds to a set of application entities, such as people, parts, projects, or organizations. The entities of a file may be from the same class (e.g., people) or from different classes (e.g., projects and organizations).

The entries of a file tend to be independent of one another, in the sense that one entry can be processed without reference to another. In the general case, however, entries of a file may be explicitly inter-related, that is, related in a manner which is known to or controlled by the system. A simple form of explicity relation is the ordering of the file entries on, say, the values of the entry sequencer. More general relations are possible by establishing nonhierarchic group relations between groups in different entries, or between the entries themselves when they are group entries.

A file whose entries are unrelated, or related only by ordering, is called an unlinked file. A file in which more general explicit relations exist among entries is called a linked file.

An unlinked file schema consists of one or more entry schemas. The usual case is that of a single entry schema. The generalization to multiple entry schemas is made for reasons similar to those for providing multiple dependent group schemas in a single group relation schema (see Figure 2-19). An instance of an unlinked file schema consists of zero or more instances of each entry schema, with the instances possibly being ordered on the basis of a sequencer which is common to the entry schemas.

A linked file schema similarly consists of one or more entry schemas. In this case, however, each entry schema may contain one or more entry schemas. In this case, however, each entry schema may contain one or more inter-entry group relation schemas, i.e., non-hierarchic group relation schemas where purpose is to establish relations among instances of the entry schema. In addition, any two group schemas from different entry schemas may be members of an inter-entry group relation schema.

An instance of a linked file schema consists of zero or more instances of each entry schema, with groups in different entry instances being related in accordance with any inter-entry group relation schemas. In addition, the entries of a file may be ordered as in the case of the unlinked file.

Figure 2-29 illustrates a linked file schema composed of a single entry schema. The entry schema is that of Figure 2-26, augmented with the inter-entry group relation schema WIFE, whose purpose is to relate pairs of entries corresponding to married couples. Figure 2-30 shows an instance of this schema.

Figure 2-31 illustrates the more complex case of a linked file schema with multiple entry schemas. The file schema is composed of four entry schemas (ORGANIZATION UNIT, PERSON, PROJECT SKILL) interconnected by six inter-entry group relations (EMPLOYS, HAS, WORKS ON, IS LEADER OF, REPORTS TO, IS RESPONSIBLE FOR). The entries in this example are group entries, although in the more general case they could be tree entries or plex entries. Figure 2-32 shows an instance of the file schema of Figure 2-31.

All systems surveyed provide the file as a data base structure. The system term for file is often qualified with a term such as "system" or "master", to distinguish the data base file from such non-data base files as transaction files and answer files (see Figure 2-33).

2.5.1 File types

Two files are of different type if they have different rules of composition or different sets of attributes.

Each of the surveyed systems provide only a single type of data base file.

2.5.2 File composition

2.5.2.1 Schema composition

Rules for file schema composition include

- restriction on number and type of entry schemas in the file schema
- restrictions on the use of inter-entry relation schemas, e.g., relations allowed between entry-defining groups only.

GIS, MARK IV, NIPS/FFS TDMS, UL/1, IMS(AP), SC-1

A file schema is composed of a single entry schema.

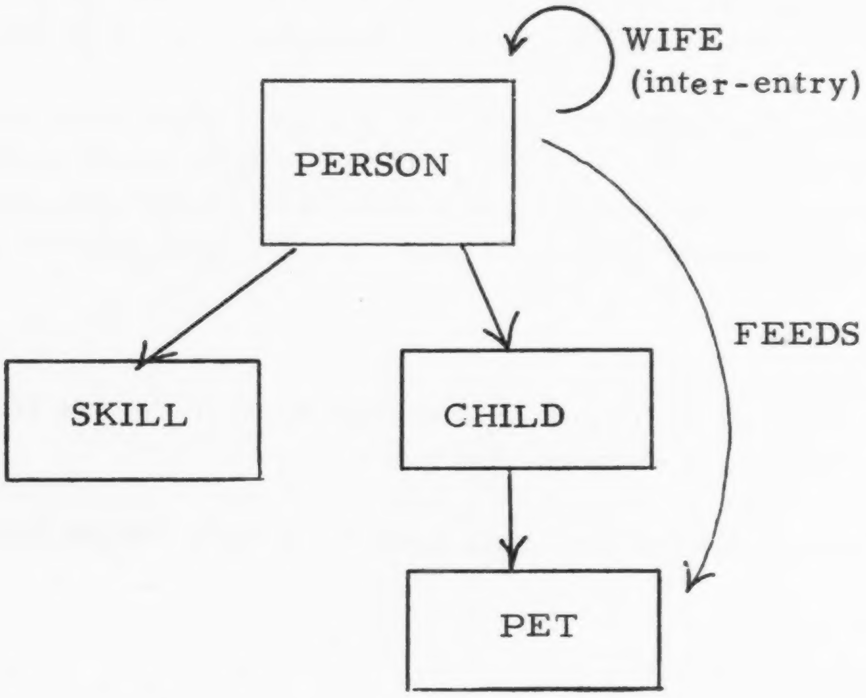

WIFE
(inter-entry)
PERSON
FEEDS
SKILL
CHILD
PET

Figure 2-29
A file schema with a single entry schema

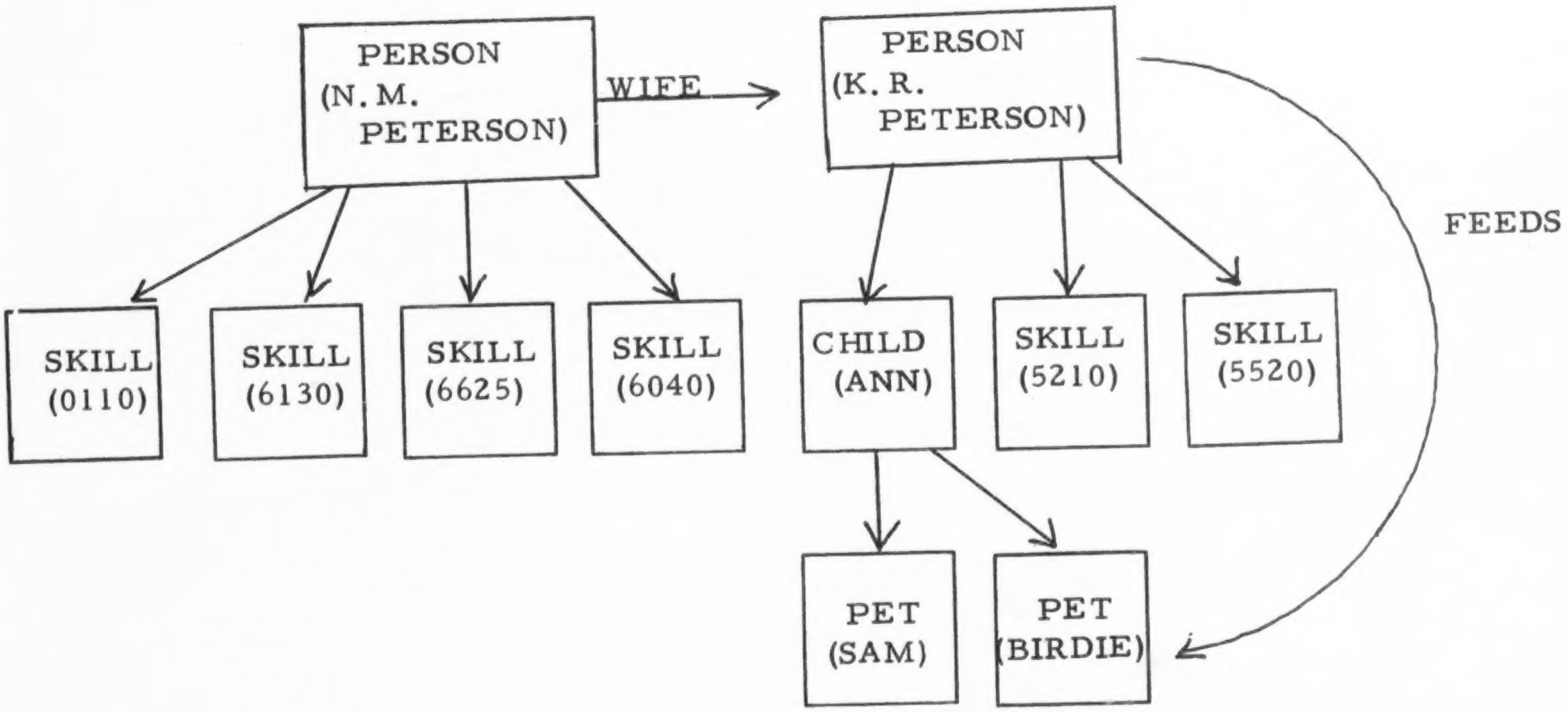

PERSON
(N.M.
PETERSON)
WIFE
PERSON
(K.R.
PETERSON)
FEEDS
SKILL
(0110)
SKILL
(6130)
SKILL
(6625)
SKILL
(6040)
CHILD
(ANN)
SKILL
(5210)
SKILL
(5520)
PET
(SAM)
PET
(BIRDIE)

Figure 2-30
An instance of the file schema
of Figure 2-29

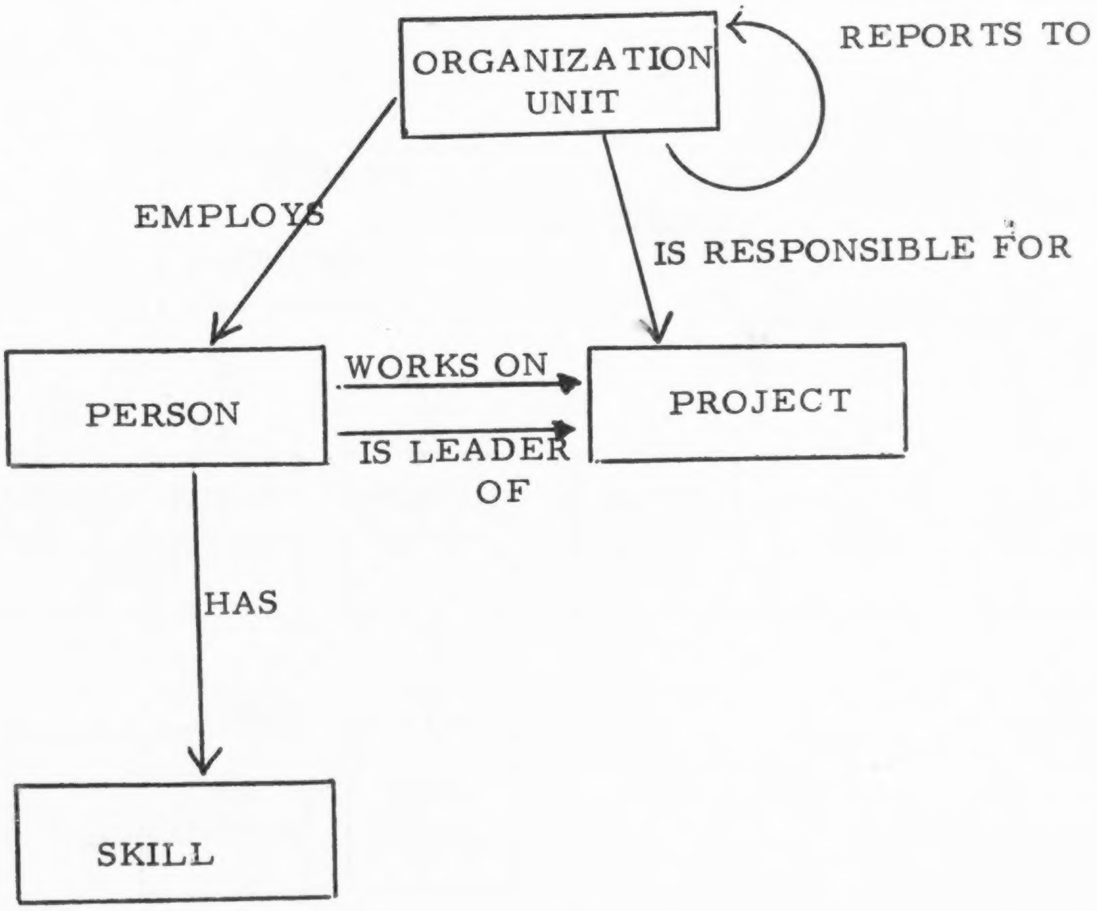

ORGANIZATION UNIT
REPORTS TO
EMPLOYS
IS RESPONSIBLE FOR
PERSON
WORKS ON
IS LEADER OF
PROJECT
HAS
SKILL

Figure 2-31
A file schema with
multiple (group) entry schemas

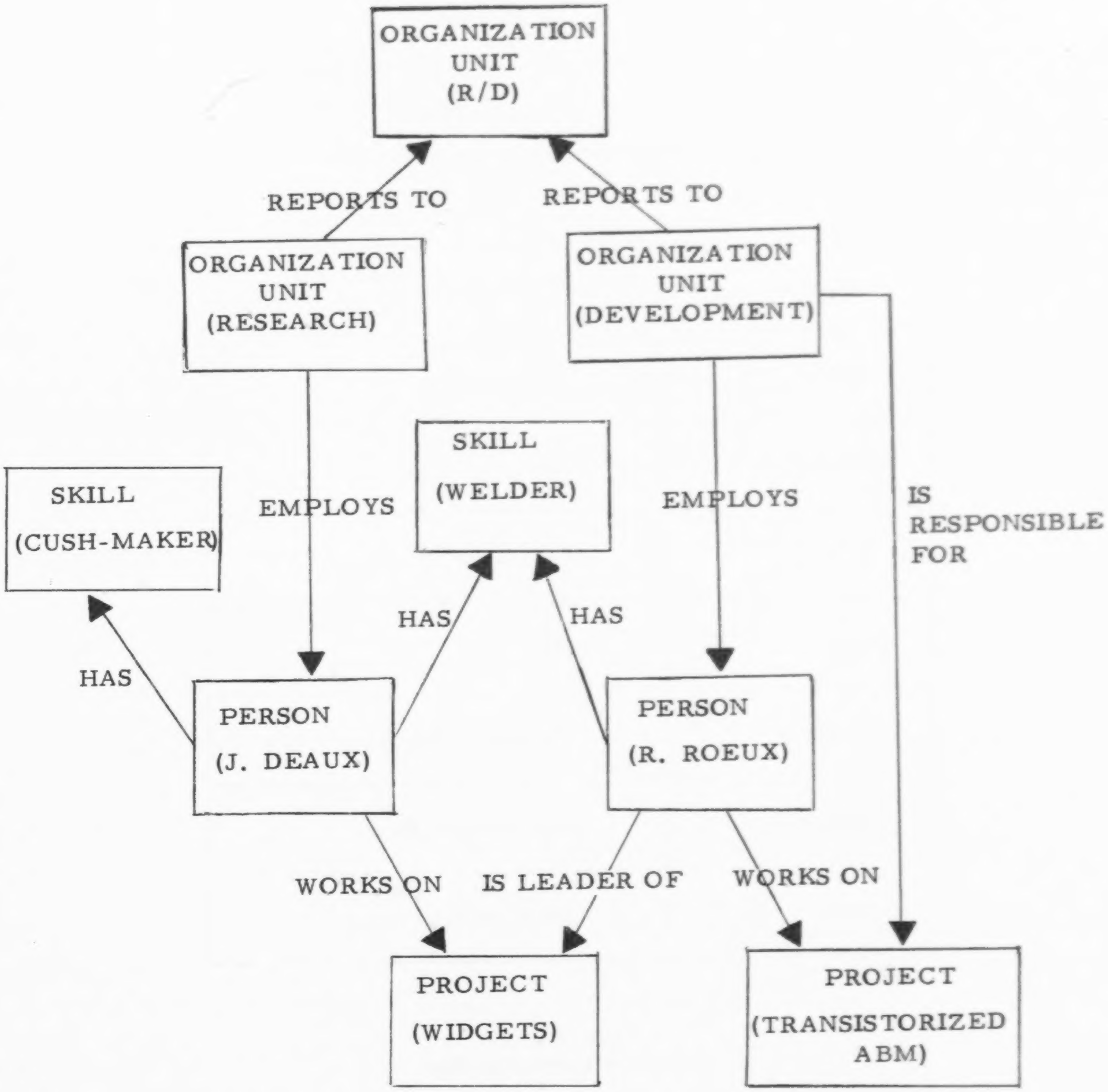

ORGANIZATION UNIT (R/D)
REPORTS TO
REPORTS TO
ORGANIZATION UNIT (RESEARCH)
ORGANIZATION UNIT (DEVELOPMENT)
SKILL (WELDER)
SKILL (CUSH-MAKER)
EMPLOYS
EMPLOYS
IS RESPONSIBLE FOR
HAS
HAS
HAS
PERSON (J. DEAUX)
PERSON (R. ROEUX)
WORKS ON
IS LEADER OF
WORKS ON
PROJECT (WIDGETS)
PROJECT (TRANSISTORIZED ABM)

Figure 2-32
An instance of the file schema
of Figure 2-31

	SYSTEM TERM
GIS	system file
MARK IV	master file
NIPS/FFS	data file
TDMS	file, data base
UL/1	master file
COBOL	file
DBTG	data base
IDS	file
IMS(DA)	physical data base
IMS(AP)	logical data base
SC-1	primary file

Figure 2-33
System's term for file

COBOL

A file schema is composed of one or more entry schemas.

DBTG, IDS

A file schema is composed of one or more group relation schemas. In IDS, the graph of the resulting structure must not contain cycles, while in DBTG there is no such restriction. Since in these systems the objects of group relations are "records", and "records" are the equivalent of entries, a file schema can be described alternatively as a set of entry schemas and a set of inter-entry group relation schemas.

In IDS, a file schema may contain at most 999 "record" schemas. "Record" schemas with no parent must have "access mode" equal to "primary" or "calculated".

A file schema may contain at most one group relation schema whose owner is the system (in IDS, the CALC chain).

IMS(DA)

A file schema is composed of a single entry schema. "Logical" group relations in the entry schema may be used for inter-entry relations.

SC-1 (AP)

A file schema is composed of a single entry schema. The entry-defining group schema may correspond to any repeating group schema within a DA entry schema, and the entry schema may consist of any subset of the structures subtended by this repeating group.

2.5.2.2 Instance composition

Composition rules for file instances include

- limitation on number of entry instances
- limitation on inter-entry relations of the type covered under group relation instances and plex entry instances.

GIS, MARK IV, NIPS/FFS, TDMS, UL/1, COBOL, IMS, SC-1

These systems employ the general rules of file instance composition cited in 2.5.

DBTG, IDS

An instance of a file schema consists of one or more instances of each group relation schema contained therein, subject to the restriction that a single group cannot be dependent on more than one parent group in the same relation.

In DBTG, the provision for file schemas with cycles implies the possibility of file instances with cycles. Such cycles can exist only when at least one "record" in the cycle has the "manual" attribute (see 2.3.3.2).

2.5.3 File attributes

2.5.3.1 Names

This includes facilities for assigning names or other identifiers to files.

All systems require a file name, as indicated in Figure 2-34.

2.5.3.2 Other schema attributes

Other schema attributes of files include

- access locks for controlling access to the file
- codes identifying the programs which make use of the file.

GIS

GIS provides access locks for query and update at the file level.

MARK IV

None.

NIPS/FFS

User may assign a security classification to a file.

TDMS, UL/1, IDS, IMS

None.

	NAME CLASS	SYNONYMS
GIS	1 to 8 characters	yes
MARK IV	1 to 8 characters	no
NIPS/FFS	1 to 7 characters	no
TDMS	1 to 255 characters	no
UL/1	1 to 8 characters	no
COBOL	1 to 30 characters	no
DBTG	1 to 30 characters	no
IDS	1 to 30 characters	no
IMS	1 to 8 characters	no
SC-1	2 to 31 characters	no

Figure 2-34
File naming

COBOL

A label record type and label contents may be associated with a file schema. Labels may be read, checked, and written with user-written procedures.

DBTG

DBTG provides access locks at the file level.

SC-1

None.

2.5.3.3 Non-schema attributes

Whenever multiple instances of a given file schema can exist at the same time, provision is usually made for assigning identifiers to the file instances. An example is a "generation number" used to designate one of a set of files with the same name.

Other non-schema attributes of a file might include

- date-time stamps
- entry counts
- control totals over file entries.

GIS

A "generation number" may be assigned to successive versions of a physical sequential (a storage structure type) system file.

MARK IV, NIPS/FFS, UL/1, COBOL, DBTG, IDS, IMS

None.

TDMS

When the file is created, the date and a "modification number" of 1 are associated with the instance. Each time the file is updated, the date is revised and the modification number is increased.

SC-1

A "generation date" of the form YY-MM-DD may be associated with versions of a primary file schema.

2.6 Data base

A data base is a set of files maintained by the system for use in user-specified creation, updating, and interrogation processes. The files of a data base are accessed through names or other identifiers supplied by the user in a prior definition process. Thus, files which are defined to the system each time they are accessed would not comprise a data base as defined here.

In the general case, the files of a data base may be inter-related. This is usually accomplished by setting up relations among the entries of different files, much in the manner that relations are established among entries of the same file (see 2.5).

A data base schema consists of one or more file schemas and possibly one or more inter-file group relation schemas whose purpose is to establish relations among entries of different files. An instance of a data base schema consists of zero or more instances of each file schema, with groups in different file instances being related in accordance with any inter-file group relation schemas.

All systems surveyed have the data base structure type as defined here.

2.6.1 Data base composition

A system's rules for composing a data base schema include

- limitations on number of file schemas accommodated
- limitations on the use of inter-file group relation schemas.

Rules for composing a data base instance include

- limitation on number of instances of each file schema
- limitations on inter-file group relations of the type covered under group relation instances and plex entry instances.

In TDMS, DBTG, and IDS, a data base schema is composed of a single file schema, and an instance of a data base schema consists of one instance of the file schema.

In all other systems, a data base schema may contain multiple file schemas. Except for GIS and SC-1, an instance of the data base schema consists of a single instance of each constituent file schema. In GIS and SC-1, a file schema may have multiple instances.

In IMS(DA), a data base schema consists of one or more file schemas and one or more "logical" group relation schemas representing inter-file relations. An instance of a data base schema consists of one instance of each constituent file schema, with files inter-related according to any inter-file relations.

2.7 Data structure generalization

Features may be provided by a system for accomodating data structures more general than those implied by the foregoing paragraphs. Examples are the establishing of explicit relationships between items of different groups (as distinct from explicit relationships between the groups as a whole); and the establishing of explicit relationships between entire entries and files (as distinct from relationships between their constituent groups).

GIS, MARK IV, NIPS/FFS, TDMS, COBOL, IMS, SC-1

None.

UL/1

UL/1 interrogation procedures may be invoked against COBOL files. To this extent, the UL/1 data structure class subsumes the COBOL data structure class.

DBTG

An additional data structure type, the "area", is superimposed on those described earlier. The "area" is a collection of "records" of various schemas. The main purpose of the "area" is to allow the data administrator to control placement of data for efficient storage and retrieval. However, the "area" is apparent to the application programmer, hence is properly regarded as data structure.

IDS

IDS provides programmer access to storage structure information stored with each record instance, thus permitting the programmer to achieve structures not within the scope of the system.

An additional data structure type, the "page", is provided in addition to those described earlier. A "page" is a subdivision of data base storage. A range of "pages" serves the same purpose as the DBTG "area".

3. DATA DEFINITION

Two fundamental objectives of generalized data base management systems are to store all user data in a data base, and to achieve independence of the data from the programs that process it. In this way data can change without necessarily causing a change in all the programs operating on it (and vice versa). Also, the centrally stored data can be used by various groups of users and still be separately managed, standardized, and protected. In order to achieve this, a unified facility is provided for the definition of the structure of the data to be stored and processed. To utilize this facility the data administrator describes the names, value classes, constituents, relationships, and all other attributes of the various data structures he wishes to establish (see Chapter 2).

Some systems allow both an overall primary data definition, and individual ones oriented toward particular users or programs, within the framework of the primary one. It also may be possible for the user to redefine his structure to a greater or less extent without completely restating the definition and restructuring the stored data.

3.1 Context of the data definition

Systems vary in the way in which data definition fits into their overall framework. In a few, the definition is an integral part of each program, concerned only with the data for that program, and complied with it. In most, however, the definition is input to the system separately, and processed to create directories or tables which are referenced in later processing steps. The definition function may be integrated with file creation (see Chapter 6), or the definition of the file and its population may be independent steps.

There are various ways of organizing the internal structure of the definition. It may be a single integrated set of statements or it may be broken into several subsections. The concept of level in a hierarchical data structure may or may not be explicitly used in the definition and the definition of elements at one level may appear before or after the definition at the next higher level. In some systems, each line of the definition contains its own identification and therefore the lines can be input in any order. In others the sequence of presentation is significant.

Finally, in different systems, data definition is done in different sorts of languages - narrative, keyword, separator, or fixed position. These types are described in the General Summary (see 1.4). Generally all data definition for a given system is done in a single language, with the details of syntax varying depending on the particular type of structure being defined. The language is often of the same form as, or closely related to, those used for interrogation, updating, and other functions.

At the end of this chapter (see 3.11), a single data structure is defined in the language of each system.

GIS

The process of data definition is combined in a separate data description task (DDT) with control of the storage allocation function of the operating system. This task is executed directly by the system's language processor, and produces tables for later reference by a compiler.

The data definition is a single free-standing set of statements in a keyword-type language. Its subsets each describe one group of a tree-structured entry, with the definition of the attributes of a group schema following the definitions of its constituent item schemas.

MARK IV

The data definition process is combined with linkage to storage allocation, and results in a set of compiled instructions
It can be done in conjunction with, or at any time prior to, file creation.

The definition is done using a printed File Definition form with fixed field positions (see Figure 3-1). File level information is entered in the heading; the form contains one or more lines for each item schema, which also incorporate group level information. Each line is internally identified, and their order of input is immaterial.

NIPS/FFS

Data definition takes the form of a separate file definition batch run. In the execution of this run, the file is established and its definition converted to a File Format Table (FFT) that is stored at the beginning of the file.

The data definition, in separator form, includes for each non-repeating group schema, the definition of its constituent item schemas, followed by the group schema definition. Repeating group schema structure is implied by codes in the item level definitions.

TDMS

Data definition is a separate function performed using the DEFINE operator. The inputs to DEFINE are narrative statements, with each statement containing an item or group schema. Statements may be entered interactively at a terminal, or handled as batch input.

The result of the DEFINE operation is a skeleton file containing all of the necessary directories, but none of the data tables which are constructed by GENERATE (see Chapter 6).

UL/1

The process of data definition is carried out by an Establishment task, which processes two parts of the data definition; Identification, which defines the item schemas only, and Structure, which sets out the group level structure. Both of these are in separator form, with the Structure section using some narrative-type elements. Other sections of the definition contain information on coded item values and on validation to be performed on item values.

If COBOL files are to be processed, the COBOL Data Division must be input (see 6.2).

COBOL

The data definition, in narrative form, is an integral part, called the Data Division, of each program processing the data, is concerned only with data for that program, and is compiled along with the procedure statements. There is no concept of data definition as a separate set of statements or independent process.

Group schema hierarchical relationships are defined by explicit level numbers, and a group schema definition precedes its item definitions.

DBTG

The system distinguishes two major data definition components:

- The schema, in which the overall data base is defined from the viewpoint of the data administrator, expressed in a Data Description Language (DDL) for the Schema.

- The sub-schema which describes the data as seen by an individual user or program, and which can be in one of a variety of languages oriented toward particular host languages. The sub-schema DDL for COBOL has been specified by the Data Base Task Group.

In this chapter, except when discussing auxiliary data definition (see 3.10), all descriptions will be only of definition as done in the Data Description Language for the Schema.

This language is narrative in form. Many of its features are devoted to expressing various attributes of group relations, and the definition of these is separated from item and group schema definition.

The data definition is a separate set of statements intended to be processed independently of any program using it.

IDS

The data definition is done within the Data Division framework of a COBOL program, with statements to define group relations being added in a narrative language. The whole program, containing the augmented Data Division, is compiled as a body.

IMS

Data definition is specified in a separator-type language and is carried out in a separate data definition job, together with linkage to the storage allocation function of the operating system. The definition is a single set of statements, with a sub-set defining each group of the tree-type entry. The group schema definition precedes those of its constituent items.

SC-1

Data structure definition is accomplished using a self-contained function which is independent of any user program and independent of any operations which populate that data structure. It takes place prior to processing, and the processed definition is stored.

The definition takes place incrementally, in that the definition process always adds (or for revision, modifies or deletes) structures to or from an existing hierarchical framework. On initial definition, that framework is empty. Later on, changes can be made at any node in the hierarchy. A repeating group schema and its instance are defined separately, at adjacent levels, followed by the definitions of its principal items and non-repeating groups, and then the definitions of any principal repeating groups. The language form is separator. A fixed position input sheet is also provided.

3.2 Item schema definition

Item names, value class and other schema attributes are defined using some form of the system's common language type (see 1.4).

The name of the item (and possible synonyms) are given. Some systems also allow or require an item number, which can be used instead of the name in the definition and elsewhere in the system. This number appears essentially as a line number in the definition.

The value class of the item may be defined by giving name or code of the item type ("BINA" for binary, "D" for date), or several of the attributes for the item value class may be combined in a "picture" statement, which contains a string of one-character codes, each defining the characters which may occur in the corresponding position of a value of that item. Editing directions may also be included (which are not part of the item value class definition). Thus "$$999V99" specifies the item value length (five decimal digits) by the 9's, the position of the decimal point by the V, and a floating dollar sign for editing by the $$.

Some systems allow the user to state a range of values which are permitted; a minimum or maximum value; or to specify, by listing them, the actual individual values which the item instance is allowed to assume. Value class attributes may be specified either at the same time as the item name and item type, or in a separate section of the date definition. Therefore, they may be considered as either an intrinsic part of the item schema definition or as part of the validation which must be applied to the data values (singly and collectively) in the data base. The user may also be able to control the item's placement (right or left justified) in an area longer than that necessary to contain it, and to set up synonyms for values.

The length of the item value may be fixed in the system, and therefore unnecessary to define, or if not fixed it may be given in digits, bytes, or characters (or be implied by a "picture" statement). The user may be required to state the starting character position of the item value in the containing group or entry, in addition to or instead of the length.

More or less extensive editing of item values, either for input/output or for validation, may be definable. This may be part of the "picture", or may be specified by giving the names of subroutines or tables which are to be used.

Security against unauthorized accesses to values is usually defined by giving one or more codes (sometimes different for query and update) which the user must supply before being allowed to use the value. In some cases the user may specify the name of a security checking procedure to be used in connection with item references.

In the case of string-type items, sub-items may be named, and defined by giving the starting and ending positions, or the length, of the sub-item within the item value.

Finally the role of the item as a group or entry identifier or sequencer may be indicated.

GIS

Item schema attributes are defined by a sequence of "keyword = identifier" clauses, separated by commas. These clauses are of the following forms:

Item name	FLD:NAME = name
Item Type	UNITS = $\left\{\begin{array}{l}\text{PACD}\\ \text{EBCD}\\ \text{BINA}\\ \text{FLPT}\end{array}\right\}\begin{array}{l}\text{Packed Decimal}\\ \text{EBCDIC}\\ \text{Binary}\\ \text{Floating Point}\end{array}$
Length	LENGTH=no.-of-bytes

An output column heading can be defined by a clause of the form:

HEADER = text

Optionally, editing and additional validation information can be given by a statement of the form:

EDIT:TYPSPC = edit-type, ERRORD = $\left\{\begin{array}{c}\text{E}\\ \text{S}\end{array}\right\}$

CONVA = value-type, LGTHA = value-length, value-statement

according to the following:

Type of editing and validation	edit-type	value-type	value-statement
Pattern	PICT	(not used)	EEDVAL = picture
Range	RNGE	limit value	EEDVAL = lower-limit, upper-limit
Set of values (explicit or in a table)	LKUP	table values	EEDVAL = value-1 [value-2]... EXTABL = external-table-name
User-supplied routine	EXIT	input value to routine	URUNT = routine-name

The type of the element shown under value-type is specified in the same way as for UNITS, and the length is given by "value-length".

ERRORD determines whether the erroneous value will (S) or will not (E) be put in the file. In either case the user is warned.

Security is specified by:

QSEC = query-access-code
USEC = update-access-code

where the access codes are arbitrary numbers in the range 0-127. The table relating access codes to users' security codes is set up by a system utility (see 8.2.2).

One or more synonyms for an item can be defined by statements of the form:

SYNM:NAME = synonym

Input and output conversion of item values is specified by:

$\begin{Bmatrix}\text{ENCD}\\\text{DECD}\end{Bmatrix}$:TYPESPC = $\begin{Bmatrix}\text{LKUP}\\\text{EXIT}\end{Bmatrix}$, edit parameters

where LKUP implies conversion through a table and EXIT implies conversion through a user-supplied routine. Additional parameters are as in EDIT.

Output formatting is given by:

MASK edit-pattern

where edit pattern is the same as for the IBM System/360 Edit instruction

Sub-items are specified by:

RDFN sub-item-name-1, length-1, [sub-item-name-2, length-2] ...

The sum of the lengths must be the same as the length of the parent item. The subitem statement must follow the FLD statement for the parent item.

MARK IV

An item schema definition is specified by filling in columns on the File Definition Form (Figure 3-1) to specify item name ("Field name"), starting byte position of item in the group ("Field location"), item length in bytes, type, number of decimal places, editing instructions, column heading, and use of the table lookup feature.

"L" in the form code identifies this line as coming from a file definition form. The digit following is zero, except for heading and table definition lines, as explained below.

Item type is given according to the following codes

P	packed decimal	F	binary
Z	zoned decimal	E	floating point
C	EBCDIC*		

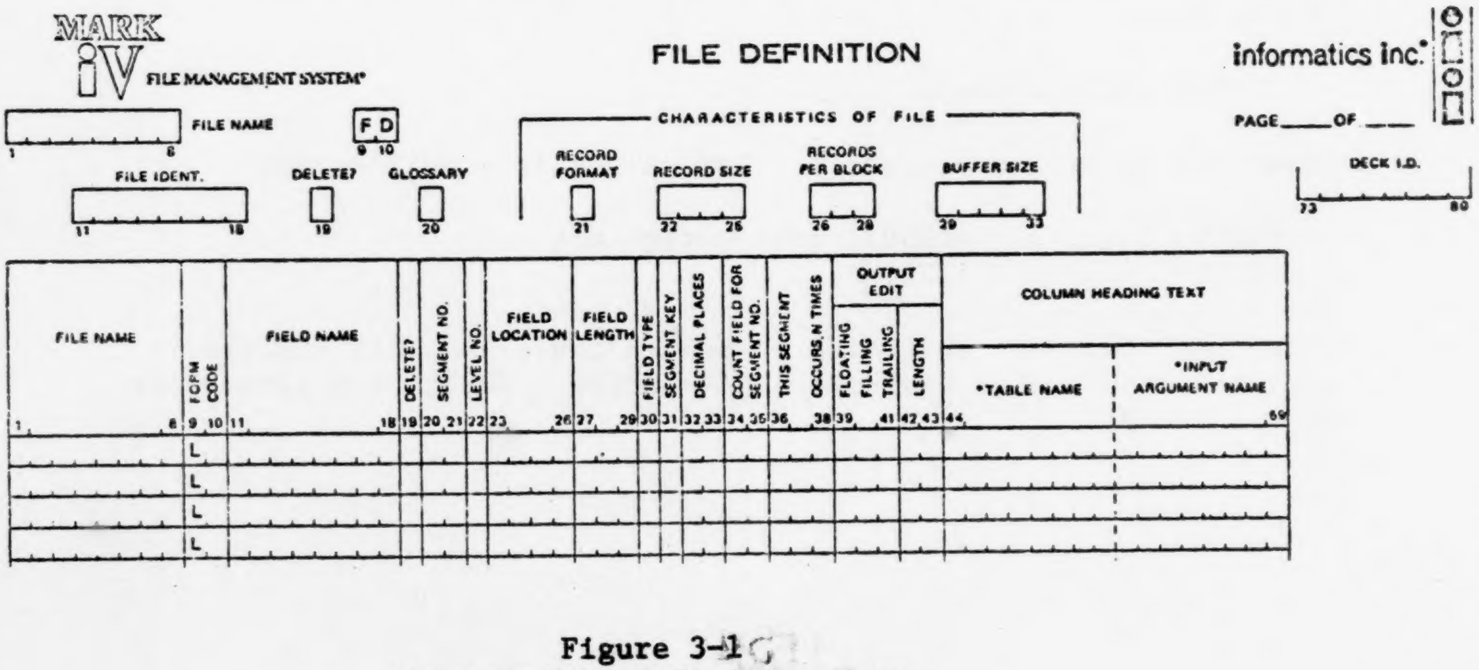

MARK IV
FILE MANAGEMENT SYSTEM®
FILE DEFINITION
informatics inc.®
FILE NAME
F D
FILE IDENT.
DELETE?
GLOSSARY
CHARACTERISTICS OF FILE
RECORD FORMAT
RECORD SIZE
RECORDS PER BLOCK
BUFFER SIZE
PAGE ___ OF ___
DECK I.D.
FILE NAME
FORM CODE
FIELD NAME
DELETE?
SEGMENT NO.
LEVEL NO.
FIELD LOCATION
FIELD LENGTH
FIELD TYPE
SEGMENT KEY
DECIMAL PLACES
COUNT FIELD FOR SEGMENT NO.
THIS SEGMENT
OCCURS N TIMES
OUTPUT EDIT
FLOATING
FILLING
TRAILING
LENGTH
COLUMN HEADING TEXT
*TABLE NAME
*INPUT ARGUMENT NAME
L
L
L
L

Figure 3-1
MARK IV File Definition Form

Floating symbols are: $ ( ) + - and trailing may be -, CR, or DB. The filling character replaces leading zeroes.

The output column heading (which can have up to nine lines) is specified in additional lines following the basic one. This is indicated, and the lines are numbered, by digits 1 to 9 (instead of 0) in the second column of "Form code".

Use of the table lookup feature is specified in two additional lines. Both have the name of the decoded value (its name as used in requests) as "Field name". The first has a "0" in "Form code" and an "R" in Field type". The second has an "A" in "Form code", the name of the table in "Table name", and the item name of the input value in "Input argument name".

NIPS/FFS

Item schema definition is done by a statement of the form

```
    FIELD  item-name  length  code  type

     [input subroutine-name]   [output subroutine-name]
     [input table-name     ]   [output table-name     ]

     [edit-mask-name]  [output-label]
```

Item length is given in characters.

The item code is C for entry identifier, X for principal item, and VX for variable length item.

Item type may be:

ALPHA*	string
NUMER	decimal numbers
COORD	geographic coordinate

The input and output subroutines or tables can be used for conversion or validation. They are not needed for COORD type items which the system converts to a special internal format. Subroutines may be written in BAL, COBOL, FORTRAN, OR PL/1.

The edit-mask may be used for output editing of numeric items. A mask may contain any character except the single quote mark that is used to enclose the mask. Except for ampersands, blanks, minus signs and CR's (credit balance symbols), the characters in the mask will print as shown. Blanks are used where source item digits are to appear; zeroes are placed where source item zeroes are not to be suppressed. The minus sign and CR are placed to the left of the first replaceable character. An alternative position for the CR is to the right of the last character.

The output-label is used only with output written at a terminal.

TDMS

The format for an item schema definition is:

```
    n  item-name      (type)      [VALUES ARE value-set-list]
                                  [FORMAT IS picture        ]
```

n is the user-chosen number given to the definition of each schema (item or group).

The number may be used anywhere in the system instead of the name, in the form Cm, for example, C4

Type may be NAME, NUMBER, or DATE.

Additional validation (beyond type) is controlled by the VALUES or FORMAT statement. In the former case "100 . . .200" indicates a range, and "100. . .$" a minimum but no maximum ("$. . .200" is also valid)

A sub-item may be defined by specifying the character positions of the subitem in the parent item, instead of an item type. Either left or right position specification may be used, for example

```
      18      BIRTH (DATE IN 11) FORMAT IS 99</>99</>99
     181      MONTH OF BIRTH (R1 ... 2 IN 18)
     182      DAY OF BIRTH (L1 ... 2 IN 18)
     183      YEAR OF BIRTH (L4 ... 5 IN 18)
```

UL/1

The item schema definition has the form:

```
#item-number      type      [item-name]       [column heading]
```

Item schemas may be referred to by number throughout the system, hence the optional item name. The optional column heading is used in output reports to identify item values.

Allowable type designations are:

ALPHANUMERIC or A
CODED or C
DATE or D
NUMERIC or N

If the type is CODED, a code table must be given in the Codes Section for each such item. It consists of a coded form and a code text. The coded form must not be more than eight characters; the code text may be up to 255 characters. Since codes are always disk resident during a run, they may fill as much space as is available.

The Code Section is followed by a validation criteria section containing statements in the selection criteria format specifying conditions on item lengths, for example,

```
#3 LENGTH (GE 6 AND LE 10) OR EQ 3 OR 12
```

Each part of an item of type DATE may be addressed separately as YEAR, MONTH, DAY; there is no other sub-item facility.

Any schema can be defined so as to have an entry date (date stamp) attached to each of its values. The set of these schemas is defined by:

```
ENTRY DATE item-name-1 [item-name-2]...
```

A multiple-valued item schema is defined by:

```
item-name  REPEATS
```

COBOL

The basic format of an item schema definition is:

```
level          item-name;  PICTURE IS picture;
                           [USAGE IS usage]
```

The level is used to define the relationship between a group schema and its constituent item schemas (see 3.3).

The "picture" is a character string in which each character is a code for the kind of character allowed or the editing to be performed at the corresponding position in an item value. Some examples are:

A	alphabetic	9	numeric
X	alphanumeric	V	(inserted) decimal point
Z	replace zero by space	$	floating dollar sign

The optional USAGE phrase specifies the data item type, as follows:

COMPUTATIONAL	Binary
COMPUTATIONAL-n	Implementor option
DISPLAY*	Unpacked (decimal if picture character is 9; alphanumeric if picture character is X)
DISPLAY-n	Implementor option
INDEX	Used as group identifier in an assembly

The significance of the numeric suffixes with COMPUTATIONAL and DISPLAY is an implementor option, which can be used to handle such things as floating point or packed numeric values, or sign type and placement.

A variable length item can be specified by:

```
PICTURE IS character-string DEPENDING ON length-value
```

where the length is specified in length-value.

Other optional elements are:

JUSTIFIED RIGHT	This causes alphanumeric values to be right justified, rather than having normal left justification.
BLANK WHEN ZERO	An item schema with this specification and having a zero value will print as spaces rather than zeros.

A multiple valued item may be defined by:

```
level  item-name  OCCURS number-1  TO number-2  TIMES
DEPENDING ON number-of-values-item
```

Two equivalent sets of values can be established for a given item schema, one coded for saving storage, and one explicit for convenience in referencing. Thus the definitions:

```
04       SALCODE  PICTURE  9
88       GRATIS   VALUE IS 0
88       LOWSAL   VALUES ARE 1,3
88       HIGHSAL  VALUES ARE 5 THRU 9
```

allow the user to state a condition as

IF LOWSAL rather than IF SALCODE = 1 OR 3

This is called a "condition name". 88 is a special level reserved for this purpose.

DBTG

The basic format of item schema definition is:

```
level  item-name   {PICTURE IS picture                             }
                   {TYPE IS type  [integer-1[,integer-2]]          }
```

The level is used to define the data structure within a group schema (see 3.3). The PCITURE feature is similar to that described earlier in this chapter. The "type" specifies the item as either BINARY or DECIMAL for FIXED, FLOATING, REAL and COMPLEX numbers. In addition, BIT, CHARACTER, DATABASE-KEY and implementor types are provided.

The values given to integer-1 and integer-2 are used to specify such attributes as the precision and scale of numeric items, and the length of string-type items.

A conversion procedure in lieu of the standard one can be specified by

```
     {ENCODING}
FOR  {DECODING}    ALWAYS CALL procedure-name
```

Security is provided for by statements of the form

```
PRIVACY LOCK FOR facility IS {procedure-name-1                  }
                             {literal-1                         }
                             {lock-name-1[VALUE IS literal-1]   }

     [OR             {procedure-name-2                  }]
     [               {literal-2                         }]
     [               {lock-name-2 [VALUE IS literal-2]  }]...
```

"Facility" can be any one or more of STORE, GET, and MODIFY. The result of the procedure name, the literal, or the contents of the lock name (which can be given an initial value by VALUE) is associated with the item, and must also appear in any program which uses any of the listed facilities on an item value.

If an item schema is part of a group schema which in turn is the parent of a dependent group schema in a group relation, then the item schema definition may contain the statement:

$$\text{IS}\ \left\{\begin{array}{l}\text{ACTUAL}\\ \text{VIRTUAL}\end{array}\right\}\quad \begin{array}{l}\text{RESULT OF procedure-name ON MEMBERS}\\ \quad\text{OF dependent-group-name}\end{array}$$

which will cause the appropriate item value to be updated (if ACTUAL) whenever a change is made in a dependent assembly. If the VIRTUAL option is used, space for the item value will not be assigned in the entry. Instead the value is calculated anew whenever a GET involving the item is performed (see 7.5.2.3).

If an item belongs to a dependent group schema, an inverse facility is invoked by

$$\text{IS}\ \left\{\begin{array}{l}\text{ACTUAL}\\ \text{VIRTUAL}\end{array}\right\}\quad \text{SOURCE IS item-name}$$

OF OWNER OF group-relation-name

which causes the value of this item to be copied, into the entry as stored if ACTUAL, or looked up at GET time if VIRTUAL, from the named item in the parent group instance.

Validity checking is specified by the statement

$$\text{CHECK IS}\ \left\|\begin{array}{l}\text{PICTURE}\\ \text{RANGE OF value-1 THRU value-2}\\ \text{procedure-name USING error-check-item}\\ \qquad\qquad\text{[,item-name] ...}\end{array}\right\|$$

which with PICTURE causes the PICTURE given previously in the item schema definition to be used for validation, as well as for the original purpose of decimal point and editing specification.

If RANGE is specified a check is made against the range whenever an item value is changed or added to the file. The checking may also be done by the named procedure, which places a zero or non-zero value in error-check-item based on the validity or non-validity of the value. The additional item names define parameters to be passed to the procedure.

Inclusion of the alerting clause:

ON [facility-list] CALL procedure-name[USING item-name-1
[item-name-2]...]

in an item schema definition results in the specified procedure being called whenever any of the listed facilities (chosen from STORE, GET, and MODIFY) is executed on a value of that item. If none are listed, the procedure will be invoked whenever the item is accessed.

IDS

Item schema definition is essentially identical to COBOL.

IMS

The format for item schema definition is

FIELD NAME = (item-name,[SEQ $\begin{bmatrix} ,U \\ ,M \end{bmatrix}$]),BYTES = length, TYPE = type,

START = position

SEQ indicates that this item schema is the group sequencer, and U or M that values are unique, or may be multiple, respectively.

TYPE is chosen from

C	Alphanumeric
P	Packed decimal
X	Binary

The value of position is starting byte of item value within group.

SC-1

The item schema is defined by a statement of the form:

level item-name type length-code LENGTH length [security]

The length of code is:

F	fixed
FO	fixed optional; value may be missing
V	variable (stated length is a maximum)

The type may be		Length specified in.
A	alphanumeric (EBCDIC)	characters
D	decimal integer	digits
E	floating point (exponential)	S (4-byte), L (8-byte)
I	binary integer	bytes
B	bit string	bits
G	general	bytes

Security may be defined by statements of the forms:

SRL	security restriction level
UAC	classa-1 [, classa-2] . . .
UMC	classm-1 [, classm-2] . . .

SRL establishes an authority level (0-15) for the item schema; a user must have been given (by the Data Administrator) an authority level at least as high, in order to access or modify the item.

The user may access (modify) the item only if some subject matter class (0-255) on his UAC (UMC) list (also set up by the Data Administrator) matches a class in the item schema's UAC (UMC) list. This allows security restriction on a "need-to-know" basis.

Both SRL and UAC/UMC checks must be satisfied before a user can gain access to a data item.

3.3 Group schema definition

The organization of the group schema definition depends on whether the group is simple or compound. If the group is simple, then each group schema definition is self-contained, usually composed of (in either order):

- A statement of the schema attributes of the group
- The definitions of the item schemas constituting the group schema.

The relationship between a group schema and its constituent item schemas may be defined explicitly, by giving a reference to the group schema in the definition of the item schema or vice versa. Alternatively, the relation may be implied by the ordering of the definitions, so that, for example, a group schema is defined to consist of all the item schemas whose definitions immediately follow (or precede) the statement which defines the attributes of the group schema itself.

On the other hand, if the group is compound, its definition must include the definitions of the contained groups. In some systems, the item and group schema definitions are interwoven, while in other systems, all of the item level information is presented separately from the definitions of the group schemas.

Group schema definition may have either a top-down or a bottom-up approach. In the top-down approach, the top levels of the hierarchy are defined first, followed by the next level and so on. In the bottom-up approach, the lowest level is defined first.

Often each element (group or item) in the hierarchy is assigned a level number which indicates how far down in the hierarchy it is. Thus in Figure 2-11, the group schema PERSON would be assigned level 1; the item schemas such as NAME and NUMBER, and the group schema SKILL, level 2; and TITLE, level 3. This allows the system to reconstruct the hierarchy from a set of sequentially-presented definitions.

The definition of a group schema which has one or more dependent (contained or subordinate) group schemas may be required to contain for each such dependent group an item schema whose value is the number of members in the corresponding dependent assembly. This is often called a count item.

Some of these characteristics of group schema definition are summarized in Figure 3-2.

	SEPARATE DEFINITION OF ITEM AND GROUP SCHEMA ATTRIBUTES		SPECIAL KEYWORD FOR GROUP		METHOD OF RELATING DEFINITIONS OF PRINCIPAL ELEMENTS TO THEIR PARENT GROUP SCHEMA		EXPLICIT LEVEL NO.	COUNT ITEM
	NON-RPTNG GROUP	RPTNG GROUP	NON-REPEATING	REPEATING	NON-REPEATING	REPEATING		
GIS	n.a.	yes	n.a.	SEGM	n.a.	precede group definition	yes (groups only)	yes
MARK IV	n.a.	no	none		n.a.	all contain group schema no.	yes (groups only)	yes
NIPS/FFS	yes	no	GROUP	none	precede group definition; names listed following group name	all contain group schema no.	no	no
TDMS	n.a.	yes	n.a.	RG or REPEATING GROUP	n.a.	"IN group-name"	no	no
UL/1	yes	yes	none	REPEATS	names listed following group name		no	no
						"UNDER group-name"		
COBOL DBTG IDS	yes		none	OCCURS	all have same level no. and follow group definition		yes (groups and items)	yes
IMS	n.a.	yes	n.a.	SEGM	n.a.	follow group definition	no	no
SC-1	yes	yes	S	F, R	all have same level no. and follow group definition		yes (groups and items)	no

Figure 3-2
Group schema definition

GIS

The definition of a group schema consists of the definitions of its constituent items, followed by a group schema definition statement of the form:

SEGM:NAME = group-name, LEVEL = level

$$\left[\text{,UNIND} = \begin{Bmatrix}\text{Y}\\ \text{N}\end{Bmatrix}\right] \left[\text{, SORT = sequencer-item-name-1, } \begin{Bmatrix}\text{A}\\ \text{D}\end{Bmatrix}, \left[\text{,sequencer-item-name-2, } \begin{Bmatrix}\text{A}\\ \text{D}\end{Bmatrix}\right]\ldots\right]$$

"Level" defines the level of the group in the entry schema: 00 for the root group schema, 01 for its dependent schema, and so on. The SORT clause specifies the group sequencer, which in turn, determines the ordering of groups in an assembly. A or D specifies ascending or descending. This phrase is optional only at the lowest level in the hierarchy of group schemas. UNIND (uniqueness indicator) is Y if the values of the sequencer item(s) uniquely identify the groups in the assembly.

A non-repeating group can be defined only by considering the elements to be sub-items (see 3.2).

The required count item is defined in the parent group in the same way as any other item schema. The dependent group schema definition must contain a statement of the form

OPTION = CNT, OPTFNM - count-item-name

where "count-item-name" is the name of the count item in the parent group schema.

A group or any portion thereof may be redefined by placing the following set of statements after the DATM Statement of a root segment or the SEGM Statement of a non-root segment.

ALTR [item-name]

item-definition-1

[item-definition-2] ...

ALTE

where each item-definition is in the form described earlier (see 3.2). The effect of this statement is to redefine the string of bytes comprising the group, starting either with the item named in the ALTR statement, or with the beginning of the group, if no item-name appears in the ALTR Statement.

The new definition need bear no relation to the original, except that the redefined group length must not exceed the original group length, and any sequencer and count items in the redefined group must have the same relative location, length, and data type as in the original.

A group may be redefined in the above manner any number of times by repeating the ALTR...ALTE sequence as many times as required. The last sequence is followed by the statement ALTR END.

MARK IV

Each item schema definition line on the file definition sheet (Figure 3-1) contains columns as follows:

Column	Function
Segment No.	Number of group schema to which the item schema belongs.
Level	Level of that group in the entry.
Segment Key	A digit from 1 to 9 which identifies the order of significance of the item schemas which are group sequencers.
This segment occurs n times	Number of occurrences per assembly, if fixed.
Count field for segment no.	Number of group schema for which this item is a count item.

Thus a group schema definition consists of all item schema definitions with the same "Segment Number".

NIPS/FFS

A non-repeating group schema is defined by:

$$\text{GROUP this-group-name} \left\{\begin{matrix}\text{item-name-1}\\ \text{group-name-1}\end{matrix}\right\} \left[\begin{matrix}\text{item-name-2}\\ \text{group-name-2}\end{matrix}\right] \cdots$$

$$\left[\begin{matrix}\text{input-subroutine-name}\\ \text{input-table-name}\end{matrix}\right] \left[\begin{matrix}\text{output-subroutine-name}\\ \text{output-table-name}\end{matrix}\right]$$

[edit-mask-name] [output-label]

In this statement the set of item and non-repeating group schemas constituting the group are listed immediately following the group name. Their definitions must have immediately preceded this one. The optional specifications of group attributes are similar to those for items, but these apply to the group as a whole as distinct from its component elements.

Repeating groups are defined by the code attribute of the constituent item schemas. The code for a repeating group is a group number starting with 1 for the first group and continuing with consecutive numbers for succeeding groups up to 255. The definition of each item schema which is part of a group schema contains as the value of "code", the number of the group schema. Thus the definition of group schema "m" consists of the set of all those item schema definitions which contain "m" in the "code" field.

The variable length non-repeating groups that can be modified, and printed or displayed are defined by a statement of the following type:

```
VSET  group-name  width-of-terminal-output-line
      [output-label]
```

Since the group has a name and does not repeat, it can be considered a variable length item. The system stores a VSET as a group and supplies it with the next repeating group number. This report also treats them as groups.

If a user wishes to designate an instance of a group (for example, for updating) by supplying the value of a group identifier, he may indicate that certain contiguous item schemas of a group schema, starting with the first one, are group identifiers by prefacing the group number with a C. If no identifier is assigned, the system assigns each instance of the group schema in an assembly (i.e. NIPS "subset") a sequential number as its identifier.

The last item of the group may be variable length. If so, its group number is prefaced by a V.

TDMS

The user assigns numbers (out of a single series) and names to item and group schemas. Either names or numbers can be used in referencing them. Group schemas are specified with a line for the group schema definition, followed by definitions for its contained item and group schemas (RG or REPEATING GROUP may be used):

```
     n    group-name-1 (RG)

 n + 1              item-name-1 (type IN group-name-1)

 n + 2              item-name-2 (type IN n)

 n + 3              group-name-2 (RG IN n)

 ...                ...
```

The indenting is only for clarity and is not required by the system. The numbers need not be sequential and subordinate items or groups can be defined at any time after their parent is defined. A non-repeating group schema must be defined as a repeating group schema which happens always to have one instance per assembly.

UL/1

Group schemas are defined in a separate STRUCTURE section, following the collection of item schema definitions which forms the IDENTIFICATION section of the file definition.

A group schema is defined in the form

```
     group-name  (item-name-1  [item-name-2]...)
```

in which the names of its constituent items are listed.

A repeating group schema has the specification

```
     group-name-1 (item-list) REPEATS  [UNDER group-name-2]
```

The VALIDATION CRITERIA section may contain a statement of the form

```
     item-name  REPEATS relational-operator integer
```

for example   SKILL REPEATS LT 6

to set limits on the size of an assembly.

COBOL

A group schema is defined by a statement of the form

```
     level group-name [OCCURS integer-1[TO integer-2 ]

          TIMES [DEPENDING ON count]]

   ┌⎧ASCENDING ⎫                                         ┐
   │⎨DESCENDING⎬      KEY IS    item-name-1[item-name-2]...│
   └⎩RANDOM    ⎭                                         ┘

    [INDEXED BY   index-name-1   [,index-name-2]...]
```

The statement is followed by one or more additional statements defining the group constituents, i.e., items or groups. "Level" defines the group's position in the hierarchy. Level numbers of constituents must be larger than that of the containing structure, but need not be consecutive.

A repeating group schema is being defined if the definition contains an OCCURS clause.

The number of groups in an assembly may be fixed or variable; in the latter case the TO phrase is included, and the value of the item "count" is the number. The sequencing and KEY statements control the order of groups in an assembly. "Index-name" is an item (which has no schema definition) whose value is used to access an individual group within an assembly.

The statement

```
     level group-name-1; REDEFINES group-name-2
```

allows two (or more) different, overlapping data structures to be established at one point in the hierarchy and consequently one area of storage to be organized and referenced according to these several different structures.

Group-name-1 and group-name-2 are each defined as described above.

This facility may <u>not</u> be used to define a file with multiple entry schemas, that is it may not be used at level 01 (see 3.6). It also may not be used in connection with groups with variable size assemblies.

An item or group synonym, or non-repeating group schema, can be defined with a statement

66 schema-name-1 RENAMES schema-name-2

[THRU schema-name-3]

If the THRU option is used, the statement defines schema-name-1 as a non-repeating group containing all the elements which are in the definition between schema-names-2 and -3. 66 is a special level reserved for RENAMES.

A group schema definition may contain a USAGE clause (see 3.2), which applies to all its constituents (none of which may be defined to have a contradictory USAGE).

DBTG

DBTG distinguishes two types of group schemas:

- The "record", which can be the object of an explicit group relation, which corresponds to the entry (see 2.4), and whose definition will be described under entry schema definition (see 3.5).
- The "data aggregate", which is a group at a level below the record.

The "data aggregate" is defined by the statement

$$\text{level group-name [OCCURS } \left\{\begin{array}{l}\text{fixed-assembly-size}\\ \text{assembly-size-item-name}\end{array}\right\} \text{ TIMES]}$$

followed by the constituent item and group schema definitions. The OCCURS phrase appears for repeating groups.

IDS

Group schema definition below the "record" level is specified in COBOL. Definition at that level is covered in entry schemas definition (see 3.5).

A security feature is provided by having a statement

AUTHORITY value

$(1 \leq \text{value} \leq 4095)$ in a principal group schema definition. This restricts all accesses to any element in the group to users who include that value in their OPEN statement.

IMS

The format for group schema definition is

SEGM:NAME = group-name, PARENT = physical-parent-group-name,
BYTES = length-of-group-instance, [FREQ =
frequency]

This is followed by the definition for the group sequencer item, and, optionally, by the definitions of other item schemas contained in the group schema. The FREQ statement, which specifies the expected number of members in an assembly, is used by the system for storage structure planning purposes.

The group schema definition also contains storage structure control statements, other group relation information, and, if this is a "logical" group schema, the definition of its "sources" (see 2.2.2.1). These aspects are covered under group relation schema definition (see 3.4) and auxiliary data definition (see 3.10).

SC-1

A non-repeating group schema definition has the form

level group-name S

followed by the definitions of its dependent data elements. (S stands for "statement", the SC-1 term for non-repeating group).

A repeating group schema is defined by a pair of statements:

$$\text{level assembly-name F [ORDERED} \begin{Bmatrix}\text{INCR}\\ \text{DECR}\end{Bmatrix} \text{sequencer-item-name-1} \left[\begin{Bmatrix}\text{INCR}\\ \text{DECR}\end{Bmatrix} \text{sequencer-item-name-2}\right] \ldots \text{]}$$

level+1 group-schema-name R

followed by the dependent element definitions.

The ORDERED phrase determines the sequence of groups in an assembly.

3.4 Group relation schema definition

The concept of group relation and its schema attributes have been described earlier (see 2.3). Among surveyed systems DBTG, IDS, and IMS provide for explicit definition of group relations. In DBTG, this definition is separate from the group schema definition; it is a part of it in IDS and IMS. In other systems which have this kind of structure, the only relation allowed is superior-to/subordinate-to, and its definition is implied by the definition of the hierarchical entry structure.

DBTG

The system name for group relation is "set". The format for its definition is:

```
SET NAME IS  relation-name   MODE IS  storage-structure-control

ORDER IS┌ [ALWAYS]  ┌FIRST┐      ┐                  ┐
        │           │LAST │      │                  │
        │           │NEXT │      │                  │
        │           └PRIOR┘      ┘                  │
        │                                           │
        └[SORTED[INDEXED [ NAME IS index name] ]    ┘
```

The ORDER statement determines whether new groups will be added to an assembly before the FIRST or after the LAST existing group, before (PRIOR) or after (NEXT) the current group (see 7.4.1), or SORTED, either indexed or as specified by:

```
┌                                   ┐
│WITHIN RECORD-NAME                 │
│BY DATABASE-KEY*                   │
│                ┌FIRST        ┐    │
│DUPLICATES ARE  │LAST         │    │
└                └NOT ALLOWED  ┘    ┘
```

If RECORD NAME, the group identifier is used; if DATABASE-KEY, the unique identifier for the group in storage is used. The DUPLICATES clause specifies whether new groups with the same identifier as one or more others in the assembly will be added before or after them, or not at all. ALWAYS indicates that the ordering of groups in an assembly is to be restored to its original state after the assembly has been re-ordered by some user-written program.

The next statement in the definition is

```
OWNER IS {parent-group-name}
         {SYSTEM           }
```

which indicates the parent group schema in the group relation, or (by SYSTEM) that there is a single assembly of this group schema in the system, whose parent is the system itself.

The OWNER statement is followed by:

```
     MEMBER IS  dependent-group-schema-name

        {MANDATORY}     {AUTOMATIC}
        {         }     {MANUAL   }
        {OPTIONAL }

        [LINKED TO OWNER  (see 9.3)]

        [DUPLICATES ARE NOT ALLOWED FOR item-name-1
          [,item-name-2]...]...
```

This statement specifies the dependent group schema in the relation. Any number of MEMBERS may be defined for one group relation.

The MANDATORY/AUTOMATIC phrase relates to insertion and deletion of groups in an assembly (see 2.3.3.2 and 7.5.3).

The DUPLICATES allows specification of a concatenation of items for which duplicate values will not be allowed.

The next clause in the statement gives the rules for selecting a particular instance of this group relation (that is, parent group and dependent assembly) for accessing of a dependent group. Its format is

```
     SET OCCURRENCE SELECTION IS THRU

    {CURRENT OF SET                                                }
    {LOCATION MODE OF OWNER [USING item-name-1[,item-name-2]...]}
```

The CURRENT option selects the current group relation instance.

If LOCATION MODE OF OWNER is specified, the selection is controlled by the LOCATION MODE clause in the definition of the parent group schema in this relation (see 3.5). If the location mode for that group is in turn via a group relation, the system may have to trace backwards through a chain of group relation schemas until one is found whose instance can be located directly.

The named items are used to select specific instances in an assembly during this process, by matching values in the instance with values in the User Working Area.

If a group schema in the chain mentioned above may be a dependent in more than one relation, a modified form of the statement allows the user to specify completely the path to be followed through the interconnected relations:

```
SET OCCURRENCE SELECTION IS THRU group-relation-name-1

USING group-relation-name-2 [,group-relation-name-3] ...
```

To determine sequencing within an assembly in conjunction with the SORTED option, the member statement may be followed by:

```
{ASCENDING* }   [RANGE] KEY IS item-name-1[,item-name-2]...
{DESCENDING }


                   [FIRST]
DUPLICATES ARE     [LAST ]  ALLOWED
                   [NOT  ]
```

The RANGE option is used in conjunction with facilities which search an assembly for a group in which an item value matches an input value. If RANGE is specified, a match will occur on the first group in the sorted assembly in which the item value is equal to or greater than the input (rather than requiring equality).

PRIVACY LOCKS and alerting can be defined for group relations, in the same way as for items (see 3.2).

IDS

A group relation is defined by placing statements of the form

```
98  group-relation-name      CHAIN MASTER
                                 [FIRST ]
                 CHAIN ORDER IS  [LAST  ]
                                 [SORTED]
```

in the definition of the parent group schema in the COBOL Data Division (one such statement for each relation of which the group schema is a parent). 98 is a special level.

The CHAIN ORDER phrase controls whether new groups will be added to an assembly before the FIRST group, after the LAST one, or according to the dependent group sequencers.

The dependent group schema definition contains the statement

```
98  group-relation-name       CHAIN DETAIL

    SELECT {CURRENT} MASTER  [MATCH-KEY IS  item-name-1]
           {UNIQUE }

           {ASCENDING }  KEY IS  item-name-2[,item-name-3]...
           {DESCENDING}

           DUPLICATES [NOT]  ALLOWED
```

If the UNIQUE option is used, MATCH-KEY specifies the item schema in the dependent group whose value determines to which parent group the dependent group is attached. Otherwise the current instance of the parent group schema is used. Item-names-2, -3, ... control sequencing in an assembly. The names are in decreasing order of significance.

The DUPLICATES clause refers to the set of item schemas in the KEY statement.

IMS

A group schema may be a physical dependent of one group schema and a logical dependent of another. This is defined by including in the group schema definition a statement of the form

```
PARENT = physical-parent-group-name, logical-parent-group-
         name
```

In order to be able to set up various sorts of auxiliary data structures (see 3.10), the group schema definition must contain storage structure control statements which will cause appropriate mechanisms to be provided (see 9.5).

Placement and ordering of groups in an assembly is determined by

```
[                                                  ] [[,FIRST]]
[RULES  = (insert-type,delete-type,replace-type    ] [[,LAST*]]
[                                                  ] [[,HERE ]]
```

where the "type" can be P (physical), L* (logical), or V (virtual) to indicate which kind of assembly with which the group is associated is to be used for insertion, deletion, or replacement. The "type" also has consequences in how the operation is performed. If a group identifier is not defined, or is not unique, the definition further can specify whether the group is to be inserted before the FIRST group in the assembly, after the LAST group, or before the current one.

3.5 Entry schema definition

A group entry schema definition usually consists solely of the union of the definitions of its constituent item and group schemas. This often causes the entry definition to have the same format as the definition of any other repeating group.

A tree or plex entry definition includes specifications of the hierarchical relationships between group schemas in the entry schema.

The definition of the relationships between groups in the entry, similarly to the definition of item-group relationships, can be done in one of the following ways:

- The group schema definition contains the name or other reference to the group schema at the next higher level in the hierarchy (This is usually omitted in the definition of a root group, for which the higher level element is the entry itself).

- The group schema definition contains an explicit level number, which defines its place in the hierarchy. In this case a definition at one level is generally followed by the definition of the elements at the next level down.

In almost all systems an entry has an identifier, often consisting of the value(s) of the identifier item(s) of the entry-defining group (see 2.4).

GIS

The hierarchical structure of the (tree) entry is defined by the level numbers in the group schema definitions; that is, a group schema has as subordinates those schemas with level number one greater than its own. (There can be only one such subordinate at any level except the lowest).

The entry identifier is composed of the set of item schemas in the root (entry-defining) group schema which have been defined to be sequencer items for that group by a "SORT = item-name,..." statement, and for which the phrase "UNIND = Y" appears.

MARK IV

The definitions of the item schemas comprising a group schema all contain the group schema number, and level, of that group. Some of the items are count items for subordinate group schemas; in that case, the column headed "Count field for segment" contains the number of the subordinate schema. The hierarchical relationship is established by the occurrence of the count item for the subordinate group among the items of the higher level group.

NIPS/FFS

An entry definition defines a two-level hierarchical entry. The definition is composed of the successive FIELD and GROUP statements that define first the "fixed set" and then successive principal repeating group schemas on the second level of the hierarchy. The definition terminates with one or more VSET statements if "Variable sets" are defined (see 3.3).

The entry identifier is indicated by the set of item schemas that contain C in the code attribute (see 3.2).

The principal items in the entry have a code attribute value of X in their schema definitions.

TDMS

A group schema contained in another has a defining statement of the form

$$\text{schema-number} \quad \text{group-name (RG IN} \quad \left\{ \begin{array}{l} \text{parent-group-name} \\ \text{parent-group-number} \end{array} \right\})$$

There is no entry identifier.

UL/1

The item schema definitions in the entry definition must be in sequence in ascending order of item number with no numbers omitted. A list of items which are date stamped may be added after all item schema definitions.

The entry identifier is taken to be the first item in the definition unless a clause

```
IDENTIFIER  item-name-1 [item-name-2]...
```

is included in the identification section.

Hierarchical relationships between group schemas in the entry are given in a separate section, called the Structure Section, using two statement types:

```
parent-group-name  (item-name-1 [item-name-2]...)

dependent-group-name  (item-name-3 [item-name-4]...)
                           REPEATS UNDER parent-group-name
```

COBOL

The entry schema is defined as a repeating group at level 01; the entry name is the defined name of that group schema.

Group schema relationships are implied by the level numbers and ordering of the group schema definitions.

DBTG

The entry name is given by

```
RECORD NAME IS  entry-name
```

Relationships of group schemas ("data aggregates") within the entry are implied by their level numbers, and the ordering of their definitions.

PRIVACY LOCKS and alerting may be defined for entry schemas in the same way as for items (see 3.2). In the entry case the facility list is larger, reflecting such operations as insert and delete, as applied to entry instances in the file.

The entry schema definition also contains accessing and storage control information (see 9.3 and 9.5).

IDS

The entry name and number are given by:

```
01  entry-name
    TYPE IS entry-schema-number
```

Group relationships within the entry are implied by level number and definition ordering. Accessing and storage control information is also given in the entry schema definition.

IMS

The entry is a simple group and consequently does not have an internal structure of group schemas.

SC-1

The entry schema definition conforms to the format for a repeating group schema definition.

Group relationships within the entry are implied by level number and ordering.

3.6 File schema definition

In all surveyed systems the definition of the attributes of the file schema is followed by the definitions of its constituent entry schemas. In most systems, only one such entry schema is allowed, resulting in the entry and file schemas being effectively equivalent and in all entries in the file having the same structure.

Some systems allow multiple logical data structures to be associated with a single physical entry type as reflected in the stored data (see 3.10).

The outline definitions below show not only the structure of the statements unique to the file schema definition, but also the sequence in which all major elements are present in the file schema definition. The indenting used is only for clarity, and does not represent system format requirements. "Group" will mean repeating group.

GIS

Security at the file level is defined by

MINFQS = query-access-code

MINFUS = update-access-code

where the access codes are arbitrary numbers in the range 0-127 which have to be associated with user security codes by a system utility (see 8.2.2). The overall structure of the definition is

```
DDT  FILE:NAME = file-name [, access-limitation] ...
     [SYN:NAME = file-name synonym]...

FLD (item schema) statements   } for the root group schema
SEGM (group schema) statement  }   ("segment")

FLD                  statements} for its dependent group schema
SEGM                 statement }
                        ...
FLD                  statements} for the last group schema
SEGM                 statement }
```

MARK IV

Spaces in the heading part of the file definition sheet (see Figure 3-1) allow specification of the file name, file identification for documentation purposes, deletion of the file rather than definition, storage structure control, and glossary output. The latter, a listing of the definitions of the elements in each entry, is according to the codes:

blank	no glossary
A	alphabetical order of element name, within segment.
1	abbreviated alphabetical order
B	location order within the file

The structure of the definition is

file-name FD

file-identification

delete-code

glossary-code

storage structure information

item schema definitions	in any order except that column heading lines must follow the corresponding item schema definition line.

NIPS/FFS

In addition to the definitions of its constituent items and groups, the file schema definition includes identification of input and output conversion tables and subroutines, and output edit masks that are referenced in any item or group schema definitions.

The file definition structure is

```
    STRUCTURE  file-name

    CLASSIFICATION  security-classification

  [[TABLE       table-name      ]      [INPUT ]   input-length
  [[SUBROUTINE  subroutine-name ]      [OUTPUT]

          output-length  input-item-type  output-item-type] ...

    [EDIT  edit-mask-name 'combination-of-blanks, zeros, ampersands,
    dashs and CR's']...

    FIELD (item schema)                         statements for the set of
         and                                    principal item schemas
    GROUP (non-repeating group schema)          ("fixed set")

    FIELD                                       statements for the first
     and                                        repeating group schema
    GROUP                                       ("variable set")
     ...                                             ...
    FIELD                                       statements for the last
     and                                        repeating group schema
    GROUP                                       ("variable set")

    [VSET  statement] ...                       for variable length non-
                                                 repeating groups

    END
```

TDMS

The outline of the file schema definition is

```
    DATA BASE NAME IS:  file-name

    TERMINATOR IS:  end-of-entry-definition-symbol

          item schema statements       for principal items

          group schema statements      for first principal group

            item and group               for elements contained
              schema statements            in first group

          group schema statements      for second principal group
                                   ...
          group schema statements      for last principal group

    end-of-entry-definition-symbol
```

Since TDMS allows missing items, multiple entry schemas ("user schemas") can effectively be incorporated into a single file by having the single entry schema which is defined to the system be composed of the aggregate of all item and group schemas in all user schemas, but arranging for any specific entry to contain values for only those item schemas belonging to a single user schema, and null values for the others.

UL/I

File schema definition, the principal function of the Establishment Division, is stated according to the following outline:

ESTABLISH file-name	
IDENTIFICATION	
item statement	for first item schema
item statement	for second item schema
...	
item statement	for last item schema
CODES	
code table statement	for item schemas
...	
code table statement	
STRUCTURE	
group statement	for first (non-repeating or repeating) group
...	
group statement	for last group
PROCEDURE	
procedure statements	required if procedures are used in validation
VALIDATION	
validation statements	in selection criteria language

Multiple user schemas can be set up by using the ability of the system to have one set of entries with values present from only one schema, and the rest of the values missing; another with values present only from a different schema; and so on.

COBOL

Multiple entry schema definitions are allowed, each beginning at level 01.

The structure of the file schema definition is

```
     DATA DIVISION

          FILE SECTION

          FD  file-name

          storage structure statements

          [;DATA RECORDS ARE entry-name-1      if there are multiple
            [,entry-name-2] ...]                 entry schemas (for
                                                 documentation only)

             01 entry-name                      for first entry schema

               02 item schema statement          for first principal item
                          ...
               02 item schema statement          for last principal item

               02 group schema statement         for first principal group

                 item and group schema             for elements contained
                   statements                        in first group
                          ...
               02 group schema statement         for last principal group
                          ...
             01 entry-name                      for second entry schema
                          ...
```

DBTG

Privacy locks for the file schema itself may be defined by a statement of the same form as that used in item schema definitions. Facilities whose use can be protected include setting of LOCKS, and DISPLAYing, COPYing, or ALTERing the schema.

The user must associate one or more areas with the file. An area represents a defined portion of the total physical data base as stored on a device; its definition is done outside of the data definition. The association is specified in the file schema definition by:

AREA NAME IS area-name [TEMPORARY]

[PRIVACY LOCK statement] ...

$[\text{ON} \begin{Bmatrix}\text{OPEN}\\ \text{CLOSE}\end{Bmatrix} \text{FOR facility-list CALL procedure-name}] \ldots$

TEMPORARY specifies that a separate copy of the area will be set up for each run unit accessing any part of the file. It disappears when the run unit terminates, and entries in it cannot be mixed in group relation instances with entries from non-TEMPORARY areas or from TEMPORARY areas of other run units.

The PRIVACY LOCK statement for "area" has the same format as for the file schema. Facilities include update, retrieval, and implementor defined support operations.

The alerting facility invoked by the ON statement causes specified procedures to be called when the area is opened or closed (which must be done by any run unit using the file) for any of the given retrieval or updating functions.

The structure of the file definition is

```
SCHEMA NAME IS  file-name

AREA NAME IS  area-name
  area statements

     RECORD NAME IS  entry-name          for first entry schema
       storage structure statements
       alerting statements
       security statements

         item schema statements            for principal items

         group schema statement            for first principal
                                             group ("data
                                             aggregate")

           item and group schema             for elements con-
                                               tained in first
                                               group
                  ...
         group schema statement            for last principal
                                             group
                  ...
     RECORD NAME IS  entry-name          for nth entry schema
                  ...
```

```
SET NAME IS group-relation-name      for first group relation
  storage structure statements         (must have been pre-
  alerting statements                  ceded by definitions
  security statements                  of groups involved in
                                       relation)

  group relation schema
    attribute statement

    OWNER group schema statements      for parent group

    MEMBER group schema statements |
              ...                  |   for dependent groups
    MEMBER group schema statements |

SET NAME IS group-relation-name      for second group relation
      ...
RECORD NAME IS entry-name            for next entry schema
      ...
```

<u>IDS</u>

The file definition is incorporated with the COBOL Data Division, and the item and group schema definition structure is much the same.

The outline of the file definition is as follows:

```
IDS SECTION

MD  file-name
storage structure statements

     01 entry-name                   for first entry schema
     storage structure statements

       item schema statements          for principal items

       group schema statement          for first principal group

         item and group schema           for contained elements
           statements
               ...
       group schema statement          for last principal group
               ...

       98 group relation schema  )
                 DETAIL statement  )     for relations in which
               ...                 )       this entry is a
       98 group relation schema  )         dependent
                 DETAIL statement  )

       98 group relation schema  )
                 MASTER statement  )     for relations in which
               ...                 )       this entry is a
       98 group relation schema  )         parent
                 MASTER statement  )
```

IMS

The structure for the file schema definition is

DBD NAME = file-name	storage structure control (also specifies if this is a logical rather than physical file)
SEGM (group schema) statement FIELD (item schema) statements	for the first group
SEGM statement FIELD statements	for the second group
...	
SEGM statement FIELD statements	for the last group
DBDGEN	
FINISH	
END	

SC-1

Initial file definition is viewed as a special case of revising the definition of the data, considered as a tree structure. Therefore if no files have been previously defined, a new file schema is added after the DATAPOOL, the (in this case, empty) node containing the files. If files already exist in the system, then the new file is added after the last old one. Whenever a structure revision is specified, the position in the tree of the element after which the new elements are being placed is defined as level zero, and level numbers of elements being added are specified relative to that one.

The structure of the definition is

ADD AFTER $\left\{\begin{array}{l}\text{DATAPOOL}\\ \text{last-file-name}\end{array}\right\}$

"F" file statement	for first file, considered as an assembly of the entry-defining group schema
"R" file schema statement	for first file schema, considered as the entry-defining group schema

```
     item schema statements              for principal items

     "F" group statement       }         for first principal group
     "R" group schema statement}           schema

       item and group schema statements    for contained elements

     ...

     "F" group statement       }         for last principal group
     "R" group schema statement}           schema

     "F" and "R" file schema             for second file
       statements
             ...

EOJS
```

3.7 Data base schema definition

Data base schema definitions consist of the collection of file schema definitions.

3.8 Processing and storage of the data definition

Input of the data definition to a system is in terms of symbolic names for the data structures, and user references to the structures in such operations as writing a program or performing an interrogation are also expressed symbolically. For storing and accessing the data, these symbolic names have to be mapped into physical storage references. Auxiliary data definition, that is, definition of the structure of the data base as seen by particular users or programs, is also in symbolic terms, and a mapping is required between the different symbolic names for the same stored values (see 3.10).

Consequently two types of mapping or binding can take place.

- Equating symbolic names in two different data definitions.

- Equating symbolic names to physical storage references.

These mappings can take place at various times, for example:

- data definition input

- program compilation

- program loading

- file opening

- value access.

In some systems the data definition is an integral part of the program, and the first two phases above are carried out in a single process of compilation. In most, however, the data definition is a separate entity, processed independently. The data definition binding varies with the stored form of the data definition. Mapping to physical storage references at data definition time often takes the form of conversion of the data definition to a table, containing, for example, relative character positions of item values, and consulted at program or process execution time.

Binding at program compilation time may result in transformation of the symbolic references in the data definition into absolute operand addresses in program instructions.

The processing of the data definition also includes converting into stored form the definitions of all defined attributes such as security, group relations, data type, etc. This information may be stored in symbolic, coded or relative form in tables, or it may be incorporated into the logic of processing programs.

In addition to storage in their converted form, user inputs such as the data definition and procedures may be stored in symbolic form by the system, in which case the definition may be accessible in that form as a data file to the user, using the normal interrogation facilities of the system.

GIS

GIS has a separate data definition task. Data definitions are input and converted to an internal form, where they are accessible to the system in compiling user procedures.

MARK IV

The data definition is input separately, is stored in a catalogue of such definitions, and is automatically accessed when the file is interrogated or updated.

NIPS/FFS

Equating of different symbolic names takes place during file redefinition (see 3.9) and during updating (see 5.3.2). The equating of symbolic names to physical storage references takes place at program compilation time.

The stored data definition is a combination of symbolic and coded information, stored in special format records at the beginning of the data file. It contains such information as the user assigned names of the data elements, their lengths and relative positions in the physical record, codes that indicate usage and type of representation, and editing and conversion specifications.

TDMS

The data definition, which is input separately, is used to set up the contents of various tables (in particular CDEFINA; see 9.3) which the system uses interpretively to access data.

UL/1

The data definition is translated into a stored data definition which the system accesses in order to find the data contained in each entry.

COBOL

The data definition is an integral part of the program, and is processed at the time the program is compiled, to be reflected in absolute addresses or their equivalents in program instructions, and in program logic.

DBTG

The "schema" defines the overall data base as seen by the data administrator (see 3.10). It is independent of any program or programming language, and is processed independently, with the method of processing and storage being defined by the implementor.

IDS

The data definition is an integral part of a COBOL program, and is represented in a data structure table which is generated at compile time, and which is used interpretively.

IMS

The data definition is processed to create a Data Base Description table which is referred to at run time to resolve symbolic references passed to it by the user program.

SC-1

The data definition process is carried out in two phases. The data structure for the whole data base, for all users, is defined, and then processed by Data Structure Definition. Each user may further set up one or more definitions of the data structure as he views it. Such an auxilliary data structure definition (see 3.10) is processed by Data Bindlist Definition. Correlation of the two definitions is actually done at file opening time (see 7.3.1.1).

The root node in the data structure is called the DATA BASE, and is a group which has a standard set of dependents. It contains the DATAPOOL (the user files) followed by all the system tables. The system tables include the stored data definition, the indexes, the dictionary of data names, physical location tables, programs, the user security list, and the auxiliary data definitions. All of these system files are stored and accessed using the normal system facilities although they are protected by high security restrictions.

3.9 Revision of the data definition

After the data structure has been defined initially, it may be necessary to modify it (add new item schemas, or change an item value length, for example). It may be necessary to re-enter the entire definition and write a procedure to restructure the stored data; alternatively input of only the changes may be required, with or without automatic reorganization of the stored data. This process of replacing an old definition with a new one should be distinguished from the facilities for setting up a definition auxiliary to the original (see 3.10).

GIS

Revision is limited to adding and deleting synonyms for item and file schemas:

```
ADD [DELETE] FILE [FIELD] schema-name
```

and to deleting an entire file schema definition.

Conversion of the stored data to a new structure, after a whole new definition is input, must be done by a user-provided procedure, which in many cases can be a GIS procedure. Parts of a previously input and stored definition can be incorporated in a new one with a CALL statement.

MARK IV

Portions of an existing file definition may be deleted or changed by specifying an item delete on the File Definition form followed by a redefinition of the item desired. Elements which can be changed are:

- Data type and length of an item
- Location of an item within a group
- Group an item belongs to
- Specification of group identifier item

The contents of the file itself are not changed. The user must write a program to do the restructuring, or it may be possible to accomplish it by using the old file as "transaction" input, and arranging for values to be copied appropriately by the system from the input to the restructured file.

An entire file definition may be deleted by specifying DELETE and the file name on the File Definition form.

NIPS/FFS

File revision can take place when the data has been defined for both the old and the new file. This means that the user must prepare a complete data definition for the new file and execute a "file structuring" run so that the data definitions for both files are available to the system in system format on direct access storage.

In setting up the new file, items in the old file may be omitted; item names, sizes, and types may be changed; and new items may be added. If items are added, no data can be put into them during file revision.

In the new file, repeating groups from the old file may be deleted or relocated but they may not be split or merged. New repeating groups may be added, but as with items, new data will not be inserted as a result of file revision.

The actual file revision takes place by executing the File Revision Processor (FR), whose minimum input is the pair of statements.

```
FILE = old-file-name
NEWFILE = new-file-name
```

In this case, data from the old file is transferred to the new file where item or repeating group names are the same. FR accepts item name changes in the form:

```
old-item-name = new-item-name
```

The user can ask for all or part of the generated file revision program to be printed or punched, and for the program to be generated, but not used to place new data in the file.

TDMS

All parts of a file definition may be changed prior to creation of the file. Changes are made by specifying use of an existing description and rewriting individual item or group descriptions (simple replacement). Item or group descriptions may also be deleted using the DELETE command. Additions are made by entering a description containing a previously unused number and name.

The data definition may be changed after creation of the file, but the file will then have to be recreated.

UL/1

The following revisions to the stored definition are possible:

- definition of new item schemas
- deletion of existing item schemas
- modification of name and type in existing item schemas

If a new item schema is defined, then a slot is created for a value in each instance of an entry. If an item schema is deleted, then the associated values in each entry instance are also removed from the file. Changes in item name allow the user to delete or replace an existing name, or to insert a name if one was not previously defined (that is, if only the required item number had been defined). Changes in item type are restricted to changes in either direction between coded items and alpanumeric items. These do not necessitate changes to the data values. A revision to the value set for a coded item also does not require access to the data.

Finally an item (either existing or new) can be defined as a repeating item, that is a repeating group consisting of one item. This does not require any change to the data, but facilitates the addition of extra values to the item using the Update function.

COBOL

Any portion of a definition may be placed in a library, given a name, and incorporated bodily into a subsequent definition by including in that definition a statement of the form

```
COPY stored definition name [REPLACING old-schema-ame
BY new-schema-name] ...
```

The REPLACING phrase allows new names to be substituted for he ones in the stored definition.

No other revision facility is provided.

DBTG,IDS

No revision facility is specified.

IMS

A new principal group schema (together with its subordinate group schemas) can be added at the end of the entry schema (that is, to the right of the last old principal group schema, looking at a tree-diagram of the data hierarchy) without having to reprogram or reorganize the stored data. If the new schema is inserted before the end, directly affected data (that is, within one data set group; see 9.5) has to be reorganized, but application programs do not need to be modified. Since the entry is tree-structured, and since the groups at a given level in the tree are processed left-to-right, if the new schema was the object of most of the processing activity, it would save running time to place it before (to the left of) groups which otherwise would not need to be accessed, rather than after them.

If a group schema now used by a program is moved to a different point in the hierarchy, either laterally or vertically, the program will probably need to be changed, since, for example, a GET NEXT (see 7.5.2.2) would refer to a different group. However, if it is possible to arrange a set of programs so that each references only the root group and one principal group schema, the latter can be rearranged to improve accessing, as described above, without disturbing the individual program.

Group schemas can be deleted from the structure, with nearly the same consequences as for insertion.

Reorganization of the stored data is handled automatically by a group of system utilities.

SC-1

Definition of the data base is done incrementally. The entire data base is hierarchical and a single execution of the data definition function is used to add a subtree to (or modify or delete) any node in the existing data structure. The first line of input to the data definition or revision function identifies the node where the new subtree is to be added.

The new subtree maybe added immediately following ("AFTER") and on the same hierarchical level as the named node. Alternatively, if a null node exists in the data structure the subtree may be added "AT" that point, that is, replace the null node. A null node is created through some previous data definition action, and is a special data structure whose only attribute is a name. It contains no values, and no dependent data structures.

The specification is done in the form

ADD $\left\{\begin{matrix}\text{AFTER}\\ \text{AT}\end{matrix}\right\}$ element-name

new-element-definition-1

[new-element-definition-2] ...

"Element-name" must be qualified by IN-phrases to resolve ambiguities. Level numbers in the new element definitions are relative, taking the level of "element-name" to be zero.

A REMOVE element-schema-name is also provided. A statement of the form

MODIFY element-schema-name-1 keyword-1 new attribute-value-1 [keyword-2 new-attribute-value-2] ...

allows the user to change any individual element attributes (e.g. length, type, etc.) in the definition.

The system keeps a central directory of file structures which is used for accessing, unless, at file open time, the system detects that the central definition has been revised since the file was last closed. In that case, to interpret the data in the file it uses a "distributed directory," which is stored with the file itself. If the file is being updated, the new version being written out will be generated under the revised definition, causing the data to be automatically rearranged to conform to the central directory. Alternatively the old file can be directly converted to the new structure by executing the "Restructure" function.

3.10 Auxiliary data structure definition

A system may provide for multiple logical data structures to be associated with a single set of stored data, each of which reflects the data as viewed by the data adminstrator, or a particular user or set of programs. Examples of differences between structures are

- There may be different names for item and group schemas
- Item value types, lengths, and other attributes may vary
- One structure may be a subset of another
- Group schemas may be organized differently.

The multiple structures may be defined at the same time, or one may be input initially, and the others added later, either separately, or each attached to an individual program. Such added definitions are supplemental to the original one; they do not revise or replace it.

The system may view all the definitions as having equal status, that is, all definitions have the same relationship to the system as a whole, or there may be a single primary one, and a number of auxiliary definitions. The auxiliary definitions are usually done in the same form as the primary data definition. In the case of equal-level definitions, the system may simply allow multiple entry schema definitions, in which case it is generally true that each of these definitions must correspond exactly to the structure of the data as stored. For instance, an item can be renamed, but if a set of item schemas is omitted, a new dummy schema having the same total length must be defined instead. Some systems, however, provide more flexibility, allowing sub-sets of item schemas to be defined and setting up automatic type and length conversion.

On the other hand, the system may provide for a single primary data definition, set up by the data adminstrator and applying to the whole data base, independent of particular users or programs; and allow each user and/or program to further define an individual data structure, within the framework provided by the principal one. This is generally found only in systems with host language facilities.

Sufficient information must be provided in such an auxiliary data definition so that data reference mapping or binding, as described earlier (see 3.8) can take place, as exemplified by the making of an explicit association between data names in a particular program and the names of their counterparts in the data base if they are different. The auxiliary definition also provides the basis on which the user program buffer area can be established (see 7.4.2).

The report of the CODASYL Data Base Task Group has outlined the purposes accomplished by an auxiliary definition, as follows:

- It need not describe the entire logical structure or the whole collection of stored data, but only those portions of concern to specific programs, in the form in which they are known to those programs.

- An individual programmer need not be concerned with the universe of the entire data base but only with those portions of the data base which are relevant to the program he is writing. Since the data base may contain data which is relevant to, and shared by, multiple applications, this may be important to ease the writing, debugging and maintaining of programs.

- A program is limited to the subset of the data that is known to it via its auxiliary definition. To a large extent, this automatically ensures the privacy and integrity of the rest of the data base from that program.

- A measure of data independence is provided for programs in that certain changes may be made to the primary definition-and the data base adjusted accordingly - without affecting existing programs using that data. This is possible because the auxiliary definition may vary in certain important aspects from the primary definition of which it is a subset.

- It allows a common language to be specified for defining a data base while allowing that part of the data base known to a program to be described in a manner which is oriented towards the conventions of the language in which that program is written.

The Data Base Task Group has also stated the following additional points regarding auxiliary definitions. These characteristics are generally present in systems offering auxiliary definition facilities.

- Object versions of primary and auxiliary definitions may be compiled independently of any user program and of each other, and stored in a library.

- An arbitrary number of auxiliary definitions may be declared on the basis of any given primary definition.

- The declaration of one auxiliary definition has no effect of the declaration of any other and they may overlap.

- A user program invokes an auxiliary definition.

- Each auxiliary definition must be named and can be invoked by an arbitrary number of programs.

- Only the part of the logical data structure included in the particular definition invoked by a program may be referenced by that program.

- Since the auxiliary definitions are host-language-oriented, a program must invoke one that is consistent with its source language.

GIS

The ALTR facility allows the stored data to be described by multiple entry schema definitions.

MARK IV

An entry can be redefined if the redefined structure corresponds exactly to the stored data.

NIPS/FFS, TDMS, UL/1

No provision is made for auxiliary definitions.

COBOL

Any number of logical structures can be applied to a single file by defining each of them as an entry. The structures all must exactly reflect the stored data.

DBTG

Full auxiliary data definition facilities are provided, in the form of the "sub-schema". So far, the data definition language for the sub-schema has been specified in the case in which COBOL is the host language. Other, future, sub-schemas will have comparable facilities. In general the sub-schema, as described in the report of the DBTG, provides the ability to:

- Select the entries or partial entries from the primary definition that are of interest to the invoking program.
- Describe the structure, format, representation and other general data characteristics of data base data in a manner that is consistent with the data description facilities of the host language.
- Establish the correspondence between primary (schema) and auxiliary (sub-schema) descriptions of the data item representations and intra-record structures.

The content of the sub-schema entry must be a sub-set of the entry described in the schema. Item schemas may be re-arranged and/or combined into group schemas to introduce additional structure into the record.

The language of the sub-schema allows specification of hierarchical record structure and data item descriptions as currently defined in COBOL. The data base data descriptions that are created using this language are similar to the traditional COBOL input-output record description entries subordinate to file descriptions (FD's).

However, the data named in each sub-schema entry is assigned locations in the invoking program's User Working Area; the data description entries, therefore, describe the format and characteristics of data as they appear in the UWA.

The following particular differences can exist between the structure as given in the primary definition and in an auxiliary definition:

- Only those elements which are defined in the auxiliary definition can be referenced by a program using this definition. All others contained in the primary definition are "removed from view". (Omission of a group also removes its constituent elements).

- Characteristics of item schemas may be different. The general rules for conversion between types are part of the specification of the data definition language. It is the responsibility of the implementor to define the correspondence between data types in the primary and auxiliary definitions. Conversion will be carried out whenever item value types differ in the schema and subschema, and it is the subject of a GET, STORE, or MODIFY function.

- Privacy locks anywhere in the structure may be different

- Ordering of elements within a higher level element may be changed.

- Auxiliary group schemas may be formed whose constituents are primary item and group schemas, and primary ones can be split.

- A repeating group schema, representing a 1-dimensional structure, may be converted into a hierarchy of group schemas, representing a multi- (in COBOL, up to 3-) dimensional structure which must preserve dimensional compatibility with the old).

- "Set selection" criteria (see 3.4) can be changed.

- New "areas" (see 3.6), or new entry accessing restrictions based on areas, can be defined.

In general the format and language for all statements in the auxiliary definition conforms to what has already been stated for DBTG, or the COBOL usage, as appropriate for the element being defined, and all facilities of either are available. Exceptions to that philosophy will be covered here, as well as statements required by the definition itself. In the case of such system functions as security, to the extent that they are not specified at a particular point in the auxiliary definition, the specification in the primary definition will be used.

The format for unique parts of the COBOL sub-schema definition is

IDENTIFICATION DIVISION

SUB-SCHEMA NAME IS sub-schema-name OF SCHEMA NAME schema-name

$$\text{[PRIVACY LOCK FOR } \begin{bmatrix} \text{LOCKS} \\ \text{DISPLAY} \\ \text{COMPILE} \\ \text{ALTER} \end{bmatrix} \text{ IS privacy-data]}$$

[PRIVACY KEY FOR COPY IS value]

The functions following PRIVACY LOCK are:

LOCKS	access item, group, etc. privacy locks
DISPLAY	view sub-schema
COMPILE	use in a program
ALTER	change

Portions of the primary definition can be copied by statements of the forms

COPY schema-name-1 [schema-name-2] ...

or

COPY ALL structure-names

Schema-name-1, schema-name-2, ... may be area, group or group relation names.

"Structure-names" can be AREAS, RECORDS, or SETS

Different set occurrence selection criteria (see 3.4) can be specified by

SET OCCURRENCE SELECTION FOR MEMBER dependent-group-name is THRU group-relation-name-1 [group-relation-name-2] ...

Name changes of elements at any level are accomplished in a RENAMING section:

structure-name NAME schema-name-1 IN SCHEMA IS CHANGED TO schema-name-2 [, schema-name-3 TO schema-name-4] ...

Structure-name can be any of the following:

AREA	area
RECORD	entry schema
DATA	item or group schema
SET	group relation schema
IMPLEMENTOR	implementor element

It must be remembered that both the primary and auxiliary definitions are describing the same stored data. This results in various restrictions, such as the maximum assembly size specified in the auxiliary definition being no larger than the corresponding one in the primary.

Item value conversion rules cover such points as the following:

- Bits 0 and 1 in bit strings correspond to characters 0 and 1 in character strings.
- Strings are extended with blanks or zeroes and truncated, on the right, with warnings or non-completion if significant characters or bits are dropped.
- Numeric values are converted according to straightforward rules, including taking into account the implications of the "picture".

IDS

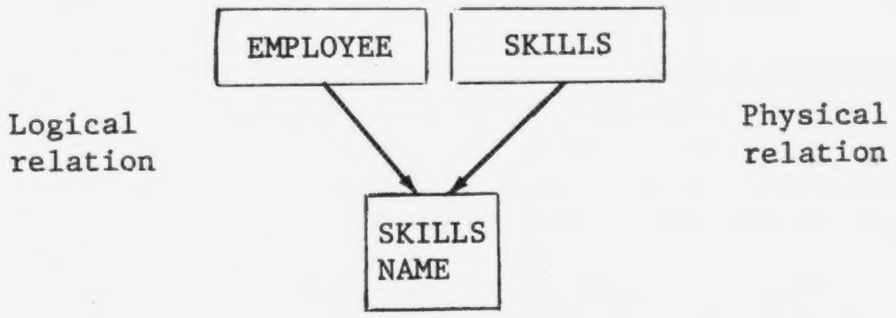
The same auxiliary definition capability is provided as in COBOL.

IMS

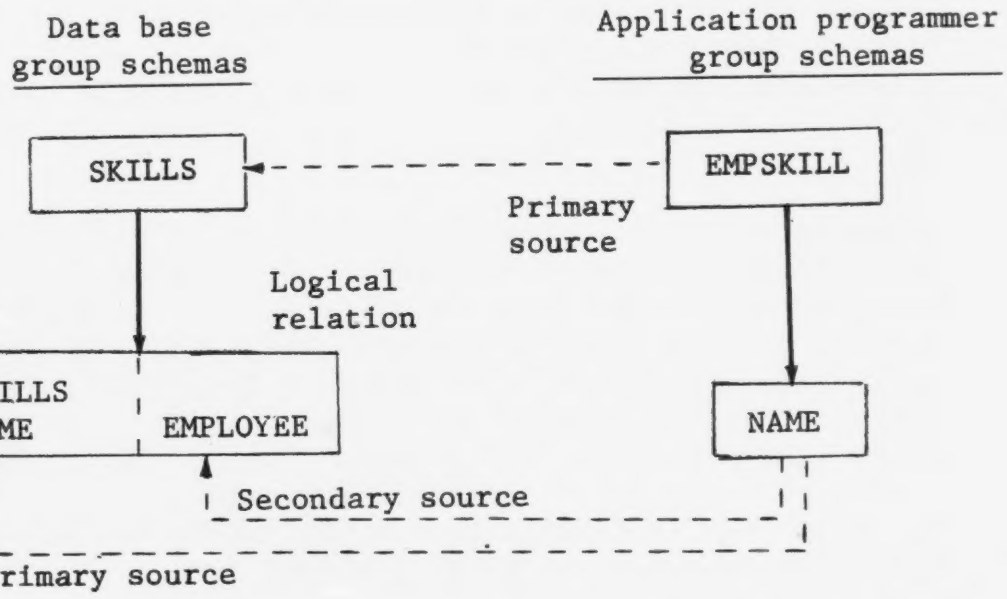
The data administrator may design the organization of the physical files ("physical data bases") (see Chapter 9) in accordance with considerations of operational efficiency; he does so keeping in mind the various user data structures ("logical data bases") that relate to this data.

The basic elements of both application programmer (AP) and data administrator (DA) data structures are the groups forming the tree-structured entries. In effect, each can set up his own set of entry schemas, with the data administrator's reflecting the data as actually stored, and each programmer's representing the data as is most convenient for his program; all the group schemas in his trees, of course, must appear somewhere in the data administrator set.

The process can be thought of as starting with the definition of the structure according to the formats already described (see Chapter 2 and 3.2 to 3.6).


EMPLOYEE
SKILLS
Logical relation
Physical relation
SKILLS NAME

The application programmer's structure can now be defined, derived from the above and also using the previously described formats:


Data base group schemas
Application programmer group schemas
SKILLS
EMPSKILL
Primary source
Logical relation
SKILLS NAME
EMPLOYEE
NAME
Secondary source
Primary source

where his EMPSKILL schema is the same as the SKILLS group schema, and his NAME group schema is the concatenation of the sources (see 2.2.2.1).

This is specified in his group schema definition by an additional statement of the form

$$\text{SOURCE = primary-source-group-name} \begin{bmatrix}\text{,KEY}\\ \text{,DATA*}\end{bmatrix} \text{[, secondary-source-group-name} \begin{bmatrix}\text{,KEY}\\ \text{,DATA*}\end{bmatrix}\text{]}$$

where KEY or DATA determines whether only the group identifier, or the whole schema, is used as a component of the group schema being defined.

The system establishes the above logical data structure essentially by setting up group relations of various kinds between groups in different physical entries, these relations reflecting the structure of the logical entry as the programmers have defined them.

In actuality the definition of each data base file contains also the definition of programmer structures associated with it.

When an application program is defined to the system, the definition contains statements of the forms

```
PCB       TYPE = DB, DBD NAME = file-name
SENSEG    NAME = root-group-name, PARENT = 0
SENSEG    NAME = group-name, PARENT = higher-level-group-name
          ...
```

where the set of SENSEG statements begins with the root group, and, working down through the hierarchy, defines all of the group schemas ("sensitive segments") to which this program is allowed access.

SC-1

After Data Structure Definition is done, a Data Bindlist Definition (DBD) function must be run before any program is executed which uses the programmer functions (see Chapter 7). A Bindlist (BL) is a directory which relates item and group names and other attributes in the auxiliary definition to those in the primary definition. It also defines sub-sets of both the data structure (schema) and the file (instances) to which this program will be bound. There may be any number of auxiliary definitions for a given file. The DBD function allows a auxiliary definition to be added, modified, or deleted. The new definition must be given a unique name .

An independent bind list may be created for a sub-tree of the data structure, conditioned by the actual stored data itself. This is done by four kinds of statements:

- ADD BL-name 01USERNAME ap-root-name

  [VERSION generation-data]

where ap-root-name specifies the application program (ap) name to be given to the root of the sub-tree being defined. Without the VERSION option, the system will use the current set of instances of the named element in what follows.

- $$\text{01 [ap-root-name]}\ \left[\text{ORDERED}\ \begin{Bmatrix}\text{INCR}\\ \text{DECR}\end{Bmatrix}\ \text{item-name-1}\ \left[\begin{Bmatrix}\text{INCR}\\ \text{DECR}\end{Bmatrix}\ \text{item-name-2}\right]\ldots\right]\text{F EQ da-root-name}$$

  [WHERE conditional-expression]

This specifies by giving its primary definition (data administrator-da) name, the node in the data structure tree at which the root of the sub-tree being defined is located. This node must be an assembly (F for "file").

The WHERE clause, if used, allows the program to be bound to a particular assembly (and its dependents) in the data, by singling out the higher level group instance in which that assembly is contained. The conditional expression (see 7.4.4) must be such as to lead to a unique such instance. (Any OR must be satisfied by only one alternative.) If WHERE is not used, then the program is bound to the whole file. For example, if the program was to be concerned only with the skills for a particular employee in a given department, the two statements might read:

```
ADD SKLYSIS 01USERNAME JOESKILL

01 JOESKILL F EQ SKILLS WHERE ORGCODE
  EQ 2340 AND EMPNO EQ 4792
```

- 02 ap-group-schema-name R - EQ da-group-schema-name
  [WITH conditional-expression]

This establishes the equivalence between the application program and data administrator names for the group schema associated with the assembly being bound. The conditional expression allows selection of a particular instance in the assembly (R for "record"). This statement might read:

```
02 JOEREC R- EQ SKILLREC WITH SKLCODE EQ 5711
```

- level $\left\{\begin{array}{l}\text{ap-name type length EQ da-name}\\ \text{FILLER W n EQ}\end{array}\right\}$

Additional statements of this form allow redefinition of name, type, and length of lower level elements in the data. The FILLER W option allows the user to specify n bytes to be skipped in the user working area, to achieve desired word alignment.

If only the sub-tree and conditional facilities are needed (i.e. no element names need to be changed), the above statements can be modified by including a phrase

$$\text{COPY}\left\{\begin{array}{l}\text{ALL da-name}\\ \text{1st-included da-name TO 1st-excluded da-name}\end{array}\right\}$$

The ALL option copies the definitions of all elements subsumed by da-name. The TO option allows, for example, a sub-set of the definitions of the item schemas in a group to be used. The included elements must be consecutive.

An existing auxiliary definition can be replaced by using REPLACE instead of ADD in the first statement above. One or more named definitions or all those associated with a given file can removed by REMOVE. Version and condition information in an existing definition can be changed by MODIFY.

3.11 Sample data definitions

This section contains a definition, according to the requirements of each system, of the simple data structure shown below. This sample is intended only to give an impression of the basic data definition, and not to exemplify all or even most system features. Also, for various reasons, the structures as defined for the different systems are not precisely the same.

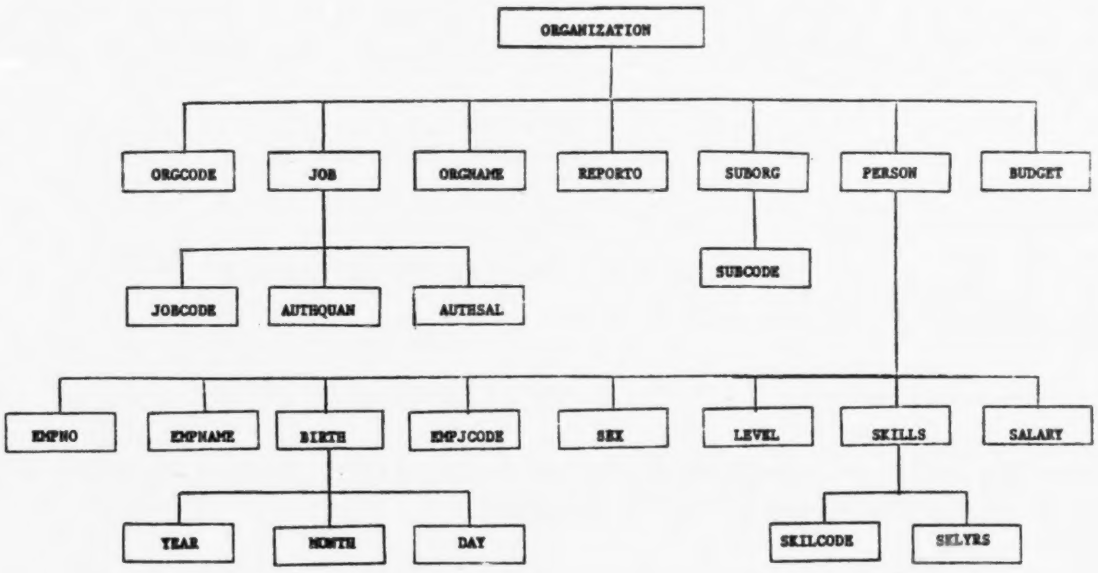

ORGANIZATION
ORGCODE
JOB
ORGNAME
REPORTO
SUBORG
PERSON
BUDGET
SUBCODE
JOBCODE
AUTHQUAN
AUTHSAL
EMPNO
EMPNAME
BIRTH
EMPJCODE
SEX
LEVEL
SKILLS
SALARY
YEAR
MONTH
DAY
SKILCODE
SKLYRS

Figure 3-3
Sample data structure

Element	Name	Type	Level
	ORGANIZATION		entry
Organization code	ORGCODE	4 digits	principal item (entry identifier)
Job	JOB		repeating group
Job code	JOBCODE	4 digits	item (group sequencer)
Authorized quantity	AUTHQUAN	4 digits	item
Authorized salary	AUTHSAL	8 digits	item
Organization name	ORGNAME	25 chars.	principal item
Higher-level org. code	REPORTO	4 digits	principal item
Sub-organization	SUBORG		repeating group (single item)
Sub-organization code	SUBCODE	4 digits	item (group sequencer)
Employee	PERSON		repeating group
Employee number	EMPNO	4 digits	item (minor group sequencer)
Employee name	EMPNAME	20 chars.	item
Birthdate	BIRTH		non-repeating group
Year	YEAR	2 digits	item
Month	MONTH	2 digits	item
Day	DAY	2 digits	item
Employee job code	EMPJCODE	4 digits	item (major group sequencer)
Sex	SEX	1 char.	item
Level	LEVEL	4 char.	item
Skills	SKILLS		repeating group
Skill code	SKILCODE	4 digits	item (group sequencer)
Years in skill	SKLYRS	2 digits	item
Salary	SALARY	8 digits	item
Budget	BUDGET	8 digits	principal item

GIS

```
DDT
FILE:NAME = ORGDATA

FLD:NAME = ORGCODE, UNITS = PACD, LENGTH = 2
   EDIT:TYPSPC = RNGE, ERRORD = E, CONVA = PACD, LGTHA = 2,
   EEDVAL = 2000, 2999
FLD:NAME = ORGNAME, UNITS = EBCD, JUST = L, LENGTH = 25
FLD:NAME = REPORTO, UNITS = PACD, LENGTH = 2, HEADER = REPORTS TO,
   EDIT:TYPSPC = RNGE, ERRORD = E, CONVA = PACD, LGTHA = 2,
   EEDVAL = 2000, 2999
FLD:NAME = BUDGET, UNITS = PACD, LENGTH = 4
FLD:NAME = NOJOBS, UNITS = PACD, LENGTH = 1
FLD:NAME = NOSUBORG, UNITS = PACD, LENGTH = 1
FLD:NAME = NOPERSON, UNITS = PACD, LENGTH = 1
SEGM:NAME = ORG, LEVEL = 0, UNIND = Y,
     SORT = ORGCODE, A

FLD:NAME = JOBCODE, UNITS = PACD, LENGTH = 2
FLD:NAME - AUTHQUAN, UNITS = PACD, LENGTH = 2,
   HEADER = AUTH. QUANT.
FLD:NAME = AUTHSAL, UNITS = FLPT, LENGTH = 4, HEADER = AUTH. SAL.
SEGM:NAME = JOB, LEVEL = 1, OPTION = CNT, OPFFNM = NOJOBS,
   UNIND = Y, SORT = JOBCODE, A

FLD:NAME = SUBCODE, UNITS = PACD, LENGTH = 2
SEGM:NAME = SUBORG, LEVEL = 1, OPTION = CNT, OPTFNM = NOSUBORG,
   UNIND = Y, SORT = SUBCODE, A

FLD:NAME = EMPNO, UNITS = PACD, LENGTH = 3
FLD:NAME = EMPNAME, UNITS = EBCD, JUST = L, LENGTH = 20
FLD:NAME = SEX, UNITS = EBCD, JUST = L, LENGTH = 1
   EDIT:TYPSPC = LKUP, ERRORD = E, CONVA = EBCD, LENGTH = 1,
   EEDVAL = M, F
FLD:NAME = EMPJCODE, UNITS = PACD, LENGTH = 2
FLD:NAME = LEVEL, UNITS = EBCD, JUST = L, LENGTH = 4,
   EDIT:TYPSPC = PICT, ERRORD = E, LGTHA = 4, EEDVAL = ZZZZ
FLD:NAME = SALARY, UNITS = FLPT, LENGTH = 4, QSEC = 102, USEC = 57
FLD:NAME = BIRTH, UNITS = EBCD, JUST = L, LENGTH = 6
   RDFN MONTH, 2, DAY, 2, YEAR, 2
FLD:NAME = SKILL1, UNITS = PACD, LENGTH = 2
FLD:NAME = SKLYRS1, UNITS = PACD, LENGTH = 1
FLD:NAME = SKILL2, UNITS = PACD, LENGTH = 2
FLD:NAME = SKLYRS2, UNITS = PACD, LENGTH = 1
SEGM:NAME = PERSON LEVEL = 1, OPTION = CNT,
   OPTFNM = NOPERSON, SORT = EMPJCODE, A, EMPNO, A
END
```

MARK IV FILE MANAGEMENT SYSTEM®

FILE DEFINITION

Informatics Inc.®

PAGE ____ OF ____

DECK I.D. (73–80)

ORGDATA. FILE NAME (1–8)   FD (9 10)

FILE IDENT. ORGDATA. (11–18)   DELETE? ☐ (19)   GLOSSARY A (20)

CHARACTERISTICS OF FILE

RECORD FORMAT V (21)   RECORD SIZE (22–25)   RECORDS PER BLOCK (26–28)   BUFFER SIZE (29–33)

FILE NAME	FORM CODE	FIELD NAME	DELETE?	SEGMENT NO.	LEVEL NO.	FIELD LOCATION	FIELD LENGTH	FIELD TYPE	SEGMENT KEY	DECIMAL PLACES	COUNT FIELD FOR SEGMENT NO.	THIS SEGMENT	OCCURS N TIMES	OUTPUT EDIT		COLUMN HEADING TEXT	
														FLOATING FILLING TRAILING	LENGTH	*TABLE NAME	*INPUT ARGUMENT NAME
1 8	9 10	11 18	19	20 21	22	23 26	27 29	30	31	32 33	34 35	36	38	39 41	42 43	44	59
ORGDATA.	L0	ORGCODE.		1	1	1	2	P	1								
	L0	ORGNAME.		1	1	3	25	C									
	L0	REPORTO.		1	1	28	2	P									
	L1	REPORTO.		1	1											REPORTS	TO
	L0	BUDGET.		1	1	30	4	P		2				$			
	L0	NOJOBS.		1	1	34	1	P			2						
	L0	NOSUBORG		1	1	35	1	P			3						
	L0	NOPERSON		1	1	36	1	P			4						
	L																
	L0	JOBCODE.		2	2	1	2	P	1								
	L0	AUTHQUAN		2	2	3	2	P									
	L1	AUTHQUAN		2	2											AUTH..	
	L2	AUTHQUAN		2	2											QUANT..	
	L0	AUTHSAL		2	2	5	4	P		2				$			
	L1	AUTHSAL		2	2											AUTH..	
	L2	AUTHSAL		2	2											SAL..	
	L																
	L0	SUBCODE.		3	2	1	2	P	1								
	L																
	L0	EMPNO.		4	2	1	3	P	2								
	L0	EMPNAME.		4	2	4	20	C									
	L0	SEX.		4	2	24	1	C									
	L0	EXPNDSEX		4	2			R									
	LA	EXPNDSEX		4	2											SEX.NAME.	SEX.
	L0	EMPJCODE		4	2	25	2	P	1								
	L0	LEVEL.		4	2	27	4	C									
	L0	SALARY.		4	2	31	4	P		2							
	L0	BIRTH.		4	2	35	6	C								▽▽/▽▽/19▽▽	
	L0	MONTH.		4	2	35	2	C									
	L0	DAY.		4	2	37	2	C									
	L0	YEAR.		4	2	39	2	C									
	L0	NOSKILLS		4	2	41	1	P			5						
	L																
	L0	SKILCODE		5	3	1	2	P	1								
	L0	SKLYRS.		5	3	3	1	P									
	L																

MK IV FORM COPYRIGHT 1969 INFORMATICS INC. ©

*USED WITH TABLE LOOKUP SPECIAL FEATURE ONLY.

LITHO IN U.S.A.

NIPS/FFS

```
STRUCTURE      ORGDATA

CLASSIFICATION                                     'UNCLASSIFIED'

TABLE    JOBSKIL   OUTPUT   4     20     NUMER    ALPHA.
TABLE    ORG       OUTPUT   4     20     ALPHA    ALPHA.

NOTE   ORGANIZATION  IS  NIPS/FFS  FIXED   SET.
FIELD  ORGCODE   4    C   ALPHA .   ORG.
FIELD  REPORTO   4    X   NUMER .
FIELD  BUDGET    7    X   NUMER .

NOTE   JOB  GROUP  IS  NIPS/FFS  PERIODIC  SET  1.
FIELD  JOBCODE   4    C1  NUMER.    JOBSKIL.
FIELD  AUTHQUAN  2    1   NUMER.
FIELD  AUTHSAL   6    1   NUMER.

NOTE   SUBORG  GROUP  IS  NIPS/FFS  PERIODIC  SET  2.
FIELD  SUBCODE   4    C2  NUMER.    ORG.

NOTE   PERSON  GROUP  IS  NIPS/FFS  PERIODIC  SET  3.
FIELD  EMPJCODE  4    C3  NUMER.    JOBSKIL.
FIELD  EMPNO     5    C3  NUMER.
FIELD  EMPLAST  22    3   ALPHA.
FIELD  EMPFIRS   3    3   ALPHA.
GROUP  EMPNAME  EMPLAST  EMPFIRS.   ALPHA.
FIELD  SEX       1    3   ALPHA.
FIELD  YEAR      2    3   NUMER.
FIELD  MONTH     2    3   NUMER.
FIELD  DAY       2    3   NUMER.
GROUP  BIRTH     BYEAR, BMONTH, BDAY.
FIELD  LEVEL     4    3   ALPHA. 'USE TO SHOW ALPHABETIC SELECTION'
FIELD  SALARY    5    3   NUMER.
FIELD  SKILL1    4    3   NUMER.
FIELD  SKLYRS1   2    3   NUMER.
FIELD  SKILL2    4    3   NUMER.
FIELD  SKLYRS2   2    3   NUMER.
```

TDMS

```
 DATA BASE NAME IS:  ORGDATA
 TERMINATOR IS:  END

 1  ORGCODE (NUMBER) VALUES ARE 2000 . . . 2999
 2  ORGNAME (NAME)
 3  REPORTO (NUMBER) VALUES ARE 2000 . . . 2999
 4  BUDGET (NUMBER)
 5  JOB (RG)
      6   JOBCODE (NUMBER IN 5) FORMAT IS 9999
      7   AUTHQUAN (NUMBER IN 5)
      8   AUTHSAL (NUMBER IN 5)
 9  SUBORG (RG)
     10   SUBCODE (NAME IN 9) VALUES ARE 2000 . . . 2999
11  PERSON (RG)
     12   EMPNO (NUMBER IN 11) FORMAT IS 09999
     13   EMPNAME (NAME IN 11)
     14   SEX (NAME IN 11) VALUES ARE M, F
     15   EMPJCODE (NAME IN 11) FORMAT IS 9999
     16   LEVEL (NAME IN 11) FORMAT IS LLLL
     17   SALARY (NUMBER IN 11)
     18   BIRTH (NAME IN 11) FORMAT IS 99</>99</>99
            181  YEAR (1...2 IN 18)
            182  MONTH (4...5 IN 18)
            183  DAY (R1...2 IN 18)
     19   SKILLS (RG IN 11)
          20 SKILL (NAME IN 19) FORMAT IS 9999
          21 SKLYRS (NUMBER IN 19)
END
```

UL/1

```
ESTABLISH  ORGDATA

IDENTIFICATION

#1  N  ORGCODE
#2  A  ORGNAME
#3  N  REPORTO           REPORTS TO
#4  N  BUDGET
#5  N  JOBCODE           JOB CODE
#6  N  AUTHQUAN          AUTH.QUANT.
#7  N  AUTHSAL           AUTH.SAL.
#8  N  SUBCODE           SUBORG. CODE
#9  N  EMPNO
#10 A  EMPNAME           EMPLOYEE NAME
#11 C  SEX
#12 N  EMPJCODE          EMPL. JOB CODE
#13 A  LEVEL
#14 N  SALARY
#15 A  MONTH
#16 A  DAY
#17 A  YEAR
#18 N  SKILL
#19 N  SKLYRS

STRUCTURE

    JOBS (EMPJCODE AUTHQUAN AUTHSAL) REPEATS
    SUBCODE REPEATS
    PERSONS (NUMBER EMPNAME  SEX EMPJCODE LEVEL SALARY
       MONTH DAY YEAR SKILL SKLYRS) REPEATS
    BIRTH (MONTH DAY YEAR)
    SKILLGR (SKILL SKLYRS)
    SKILLGR REPEATS UNDER PERSONS

CODES
    SEX M MALE F FEMALE
```

COBOL

```
DATA DIVISION

FILE SECTION

FD  SDCDATA

01       ORG
  02       ORGCODE ; PICTURE IS 9999
  02       ORGNAME ; PICTURE IS A(25)
  02       REPORTO ; PICTURE IS 9999
  02       BUDGET ; PICTURE IS Z(8); USAGE IS COMPUTATIONAL-1
  02       JOB ; OCCURS 1 TO 50 TIMES ASCENDING KEY IS JOBCODE
    03       JOBCODE ; PICTURE IS 9999
    03       AUTHQUAN ; PICTURE IS 99; USAGE IS COMPUTATIONAL
    03       AUTHSAL ; PIC ZZZZZZ ; USAGE-COMP-1
  02       SUBORG ; OCCURS 0 TO 20 TIMES ASCENDING KEY IS SUBCODE
    03       SUBCODE ; PIC 9999
  02       PERSON ; OCCURS 1 TO 999 TIMES ASCENDING KEYS ARE
             EMPJCODE, EMPNO
    03     EMPNO ; PIC Z9999
    03     EMPNAME ; PIC A(20)
    03     SEX ; PIC A
    03     EMPJCODE ; PIC 9999
    03     LEVEL ; PIC AAAA
    03     SALARY ; PIC ZZZZZ ; USAGE COMP-1
    03     BIRTH
      04     MONTH ; PIC 99
      04     DAY ; PIC 99
      04     YEAR ; PIC 99
    03     SKILLS ; OCCURS 1 TO 9 TIMES ASCENDING KEY IS SKILCODE
      04     SKILCODE ; PIC 9999
      04     SKLYRS ; PIC 99
```

DBTG

```
SCHEMA NAME IS ORGDATA

AREA NAME IS ORGPART

RECORD NAME IS ORG

PRIVACY LOCK IS SESAME

   01     ORGCODE  PICTURE IS "9(4)"
   01     ORGNAME  TYPE IS CHARACTER 25
   01     REPORTO  PICTURE IS "9999"
   01     BUDGET  TYPE DECIMAL FLOAT ; IS ACTUAL RESULT OF SALSUM
            ON MEMBERS OF PERSONS
   01     NOSUBORG  TYPE BINARY
   01     SUBORG OCCURS NOSUBORG TIMES
     02     SUBCODE PICTURE "9999"
RECORD NAME IS JOB
   01     JOBCODE  PICTURE "9999"
   01     AUTHQUAN  PICTURE "99"
   01     AUTHSAL  TYPE FLOAT
RECORD NAME IS PERSON
   01     EMPNO  PICTURE "9(5)"
   01     EMPNAME  TYPE CHARACTER 20
   01     SEX  PICTURE "A"
   01     EMPJCODE  PICTURE "9999"
   01     LEVEL  PICTURE "X(4)"
   01     SALARY  PICTURE "9(5)V99" ; PRIVACY LOCK FOR GET IS
            PROCEDURE AUTHENT
   01     BIRTH
     02     MONTH  PICTURE "99"
     02     DAY  PICTURE "99"
     02     YEAR  PICTURE "99"
   01     NOSKILLS  TYPE BINARY
   01     SKILLS OCCURS NOSKILLS TIMES
     02     SKILCODE  PICTURE "9999"
     02     SKLYRS  PICTURE "99"
SET NAME IS JOBS; ORDER IS SORTED
  OWNER IS ORG
  MEMBER IS JOB OPTIONAL AUTOMATIC; ASCENDING KEY IS
    JOBCODE DUPLICATES NOT ALLOWED
SET NAME IS PERSONS; ORDER IS SORTED
  OWNER IS ORG
  MEMBER IS PERSON OPTIONAL AUTOMATIC ; ASCENDING KEY IS
    EMPJCODE,EMPNO DUPLICATES NOT ALLOWED
```

IDS

```
DATA DIVISION
IDS SECTION
MD ORGDATA

 01    ORG               TYPE IS 100 ; AUTHORITY 2192

   02    ORGCODE ; PIC 9999
   02    ORGNAME ; PIC A(25)
   02    REPORTO ; PIC 9999
   02    BUDGET ; PIC Z(8)
   02    SUBORG OCCURS 0 TO 20 TIMES ASCENDING KEY IS SUBCODE
     03    SUBCODE ; PIC 9999

   98  CALC CHAIN DETAIL RANDOMIZE ON ORGCODE
   98  JOBS CHAIN MASTER CHAIN-ORDER IS SORTED
   98  PERSONS CHAIN MASTER CHAIN-ORDER IS SORTED

 01    JOB               TYPE IS 101

   02    JOBCODE ; PIC 9999
   02    AUTHQUAN ; PIC 99
   02    AUTHSAL ; PIC Z(6)

   98  JOBS CHAIN DETAIL
         SELECT CURRENT MASTER
         ASCENDING KEY IS JOBCODE
         DUPLICATES NOT ALLOWED

 01    PERSON            TYPE IS 102

   02    EMPNO ; PIC Z9999
   02    EMPNAME ; PIC A(20)
   02    SEX ; PIC A
   02    EMPJCODE ; PIC 9999
   02    LEVEL ; PIC AAAA
   02    SALARY ; PIC ZZZZZ
   02    BIRTH
     03    MONTH ; PIC 99
     03    DAY ; PIC 99
     03    YEAR ; PIC 99
   02    SKILLS OCCURS 1 TO 9 TIMES
     03    SKILCODE ; PIC 9999
     03    SKLYRS ; PIC 99

   98    PERSONS CHAIN DETAIL
           SELECT CURRENT MASTER
           ASCENDING KEY IS EMPJCODE,EMPNO
           DUPLICATES NOT ALLOWED
```

IMS

```
DBD       NAME = ORGDATA

SEGM      NAME = ORG, BYTES = 38, START = 1
FIELD     NAME = (ORGCODE SEQ, U), BYTES = 2, TYPE = P
FIELD     NAME = ORGNAME, BYTES = 25, START = 3, TYPE = C
FIELD     NAME = REPORTO, BYTES = 2, START = 28, TYPE = P
FIELD     NAME = BUDGET, BYTES = 4, START = 30, TYPE = P

SEGM      NAME = JOB, BYTES = 8, FREQ = 31, PARENT = ORG
FIELD     NAME = (JOBCODE, SEQ, U), BYTES = 2, TYPE = P

SEGM      NAME = SUBORG, BYTES = 2, PARENT = ORG
FIELD     NAME = (SUBCODE, SEQ, U), BYTES = 2, TYPE = P

SEGM      NAME = PERSON, BYTES = 40, PARENT = ORG
FIELD     NAME = (EMPJCODE, SEQ, M), BYTES = 2, TYPE = P
FIELD     NAME = (EMPNO, SEQ, U), BYTES = 3, TYPE = P

SEGM      NAME = SKILLS, BYTES = 3, PARENT = PERSON
FIELD     NAME = (SKILCODE, SEQ, U), BYTES = 2, TYPE = P

DBDGEN

FINISH

END
```

SC-1

```
ADD AFTER DATAPOOL

01    ORGFILE       F    ORDERED INCR ORGCODE ; ;
02    ORGREC        R    ;
03    ORGCODE       D    F LENGTH 4 ;
03    ORGNAME       A    V LENGTH 25 ;
03    REPORTO       D    FO LENGTH 4 ;
03    BUDGET        E    LENGTH S ;

03    JOBS          F    ORDERED INCR JOBCODE ;
04    JOBREC        R    ;
05    JOBCODE       D    F LENGTH 4 ;
05    AUTHQUAN      D    F LENGTH 4 ;
05    AUTHSAL       E    LENGTH S ;

03    SUBORGS       F    ORDERED INCR SUBCODE ;
04    SUBORGREC     R    ;
05    SUBCODE       D    F LENGTH 4 ;

03    PERSONS       F    ORDERED INCR EMPJCODE,EMPNO ;
04    PERSREC       R    ;
05    EMPNO         D    F LENGTH 6 ;
05    EMPNAME       A    V LENGTH 20 ;
05    SEX           A    F LENGTH 1 ;
05    EMPJCODE      D    F LENGTH 4 ;
05    LEVEL         A    F LENGTH 4 ;
05    SALARY        E    LENGTH S SRL 02 UAC 150 UMC 124 ;

05    BIRTH         S    ;
06    MONTH         D    F LENGTH 2 ;
06    DAY           D    F LENGTH 2 ;
06    YEAR          D    F LENGTH 2 ;

05    SKILLS        F    ORDERED INCR SKILCODE ;
06    SKILREC       R    ;
07    SKILCODE      D    F LENGTH 4 ;
07    SKLYRS        D    F LENGTH 2 ;.

EOJS
```

4. INTERROGATION FUNCTION

Interrogation is the selection of data from a data base, extracting (namely copying) the selected data, possibly for intermediate processing such as sorting or summarization and formatting the results either into readable reports or into a machine readable form for further processing.

For the interrogation function, information is usually entered initially to identify the part of the data base on which the interrogation is to be performed. This is followed by a statement of the selection criterion (the premise) and a statement of the extraction (the action). Identifying the part of the data base could in effect be considered part of the selection criterion, although it is normally given separate treatment.

In a self-contained system, an interrogation is a premise action grouping where both act on all or part of a file or, if inter-file relationships are possible, on a data base. Various complexities of premise action grouping are possible with the simplest case having one premise and one action. These are separated in this chapter into conditional expressions and extraction, although in some systems it is difficult to make such a clear separation.

In the host language systems, the premise usually acts on a single stored entry or on some part thereof. The resulting action may initiate a data transfer from one level of memory to another, and an unlimited complexity of premise action groupings is permitted in the host language.

The most significant difference between self-contained and host language systems is in the handling of premise action groupings. In the case of self-contained systems, the user specifies the information he requires while the process by which it is obtained is built into the system. In a host language system, the user usually programs (in the conventional sense) the series of premises and actions which define the process by which the information required is to be generated. In this chapter the functions described are those normally appropriate to a self-contained system. However, in some host language systems, facilities may be provided (in addition to those already in the host language) for expressing conditional expressions on the data in an entry. These may be of the same complexity as in a self-contained system. In such cases the capability is covered in this chapter under conditional expressions.

GIS

Interrogation is specified using a QUERY sub-procedure which must be part of a procedural task specification. The reading of files and the selection of entries is specified using a LOCATE-EXHAUST block in which the selection of entries is embedded. A further similar block of statements can be nested within the principal block to select repeating group instances within the entry. Inside any such block, the user must specify whether data is to be extracted (that is, printed) when found or whether it is to be stored in a hold file for possible sorting or other further processing before finally being printed.

Most statement types may be labelled and a transfer (or GO TO) statement is provided. Up to 99 temporary numeric items and up to 99 string items may be used in a procedural task specification for storing data values which are needed in the course of executing the procedure. Also up to 99 special TALLY items may be defined for the purpose of developing counts.

Interrogation must therefore be programmed and the user has the ability to define his own processing algorithm. The QUERY sub-procedure may access up to 16 temporary work files, called hold files, and system files.

MARK IV

Interrogation is performed according to built-in search processing algorithms, the choice of which depends on file level storage structure. Two levels of user specifications are provided, each using its own forms printed for input preparation. For simple interrogations a form is used which allows specification of the conditional expressions on the data in one file only and uses a format for the content of the output which is largely system defined, although the user may exercise some control. Sorting may be specified using this form.

For more complex interrogations, the Processing and Record Selection form must be used together with up to three associated forms which allow specification of more elaborate report formats and also the interrogation of a set of coordinated files. As part of the selection definition, arithmetic operations may be performed on numeric data and logical operations may be performed on any type of data item.

Temporary items may be defined to facilitate computation of numeric quantities, for storing constants and for communication between successive entries during processing.

A link to subroutines written in procedural languages such as COBOL and FORTRAN is provided. This provides capability for specifying processing which cannot be handled by the system within the framework of the built-in processing algorithm.

NIPS/FFS

Three different system facilities may be used for interrogation. The most complete extraction capabilities are provided in the batch mode by the Retrieval and Sort Processor (RASP). It produces a Qualifying Data File (QDF) and a Qualifying Record Table (QRT) as output.

A second facility, which also operates in the batch mode, is called the Output Processor (OP). It can extract data from data files or from the output of RASP. The extracted data can be written on disc, magnetic tape, punched in cards, or listed on a printer. A simple form of report is provided in the default options of the report specifying statements, that can also be used to specify both content and format in detail.

The Quick Inquiry Processor (QUIP) is the third facility. It operates in either the batch or on-line mode. As with OP, data can be extracted from either data files, or from RASP output. In the batch mode, data may be extracted from either sequential or indexed sequential files. In the on-line mode, data can only be extracted from indexed sequential files. The extraction may be performed either by initiating execution of a stored query or by entering a QUIP query at the terminal. Output from QUIP is formatted either as one page of infinite length suitable for terminal display or as pages with a specified number of lines suitable for listing.

When RASP is used on one file, multiple IF statements may be used to create multiple answer sets. The RASP sort statement can then generate pointers in the QRT to sort each answer set on data items and literals specified independently for each answer set by the user. When data is extracted from multiple files (up to 10), only one retrieval may be processed at a time. The data extracted from the multiple files is merged by sorting after it has been extracted. The merged file can then be printed by using OP.

Arithmetic capabilities are available through the use of subroutines that can be written in COBOL, FORTRAN, or S/360 Assembler Language and in the report summarizing operators of OP and QUIP such as TOTAL, COUNT, COMPUTE and VARIANCE. For OP and QUIP queries specified at a terminal, data may be extracted from only one file at a time.

TDMS

Interrogation is designed for on-line interactive and also for off-line use. In both cases the processing algorithm is built into the system and uses the set of secondary indexes which are automatically created and maintained for all data items in the entry.

Two levels of interrogation are provided. The simpler one, called QUERY, can only be used on-line. The other, called COMPOSE, may be used either on-line or off-line. COMPOSE includes facilities for sorting the output data, and this is performed by a built-in sort on index values.

As part of the QUERY facility the user may have the file schema displayed at the terminal and may also browse through his file gradually converging on the information he is seeking. The output generated by QUERY is in a format defined by the system.

COMPOSE has full facilities for formatting a report. It is also possible to prepare a report format interactively from the terminal.

Facilities for embedding arithmetic computation are provided but the language is completely non-procedural with no statement labels. A range of trigonometric and statistical functions may be invoked.

UL/1

Interrogation is specified in the interrogation division using a facility called the criterion language to specify a conditional expression and one called the publish section to specify the data to be extracted for inclusion in reports or sub-files.

The publish section has two levels, one in which the format of the content data in the report or sub-file is provided by the system and the other in which the report or sub-file has the format of the content data specified in detail by the user.

Processing algorithms are provided by the system. When a set of interrogations is entered, then the file (or coordinated set of files) is searched sequentially.

Sorting is performed on items or on computationally derived quantities selected from the entry as soon as it is determined that the entry satisfies the selection criterion.

No statement labelling or transfer statements are provided, and arithmetic computation may be specified using a component called the procedure language. Procedures may be invoked in the conditional expression and in the publish section.

COBOL

Interrogation is specified in a use of the Procedure Division. The reading of files and the selection of entries is specified using READ file statements and IF statements. Since COBOL is a procedural language, the file processing algorithm and the extraction specification must be programmed as a mix of premise and action statements. If the data extracted is to be included in a report, then the report writer facility may be used.

If the entries to be selected are also to be sorted, then the SORT facility may be used. The user must use temporary work files where necessary and is also responsible for management of his own working areas within high speed memory.

SC-1

Interrogation is specified by using Report Data Compiler (RDC-1) statements which indicate the data items to be used in a report and the format of the report. A group of statements, called a Report Description, defines the formatting of the items from one entry or a repeating group (and any manipulation of those items). Within each Report Description the statements are usually divided into a non-procedural section, which defines such things as headers, margins, and page length, and a procedural section which generates the detail lines for each entry. The procedural statements may be labeled and a control transfer statement is provided. Procedural statements within a Report Description may also call other Report Descriptions which format or manipulate data from other repeating groups.

Report Description programs are generated by the RDC Page Compiler from the user-written statements. A table, called a "page catalog", contains the compiled Report Description programs. The RDC Report Generator retrieves the Report Description stored in the page catalog by the compiler to generate a specified report.

Temporary items may be used for arithmetic computation including summation. A link to user-specified subroutines in FORTRAN, COBOL, or Assembler is also provided for more complex computations.

The file processing algorithm is automatically determined by the specifications of repeating groups used in the Report Description and by the statements used to call other Report Descriptions.

4.1 Characteristics of premise action relationship

The permitted premise action mix, and the nature of the actions permitted, are together a measure of the procedurality of a system. The premise action mix is the combination of premises (or conditional statements) and action statements allowed in a use of a function. The premise may be quite complex in itself and is covered in the discussion on conditional expressions (see 4.3). The actions may be of varying types and complexity depending on the system.

If a complete mix is permitted and the actions permitted include moves, data transfers, assignment statements and jump statements, then the language is essentially a procedural language such as COBOL. If the mix is restricted and the actions permitted are high level ones, which imply the processing of a whole file or even of a data base, then the language is less procedural.

GIS

Most statements types may be labeled and a transfer statement is provided allowing a fairly complete premise action mix. The interrogation function must be specified using a special set of statements called a QUERY sub-procedure within which other specific statements must be used to define the processing algorithm.

MARK IV

The premise action mix in the interrogation function is limited by the form chosen. A more complex mix is possible with the Processing and Record Selection form than with the Information Request form.

NIPS/FFS

The premise action mix in the interrogation function is built-in for the part of the functions which causes either limited or complete passing of the interrogated file or files. A more complex mix is permitted for sorting the file of entries which satisfy the selection criteria. None of the three facilities use statement labeling or transfer statements.

TDMS

The premise action mix in the whole interrogation function is limited by a built-in processing algorithm.

UL/1

The premise action mix on the whole interrogation function is limited on the top level by the built-in processing algorithms. If computational procedures are invoked, a more complex mix is allowed within the procedure definition, but no statement labels or transfer statements are provided.

COBOL

A set of statements comprise a section of a paragraph in the Procedure Division. Either of these may be given a procedure name and transfer statements contain a procedure name. Conditional statements have both a true and a false path. With these facilities, the premise action mix may be of unlimited complexity.

SC-1

The premise action mix is restricted by the file processing algorithm to the processing of a whole file, but a complex premise action mix is permitted for each entry in the file since the statements within each report description are procedural.

4.1.1 Procedural language actions

Procedural language actions are covered under Programmer Facilities (see Chapter 7). They include statements to open and close files, cause transfers between levels of memory, move data from one working area in high speed memory to another and to equate a numeric data item to an arithmetic function of other numeric items. Statement labeling is provided and associated statements to transfer execution control. In addition looping statements are provided to cause an action or set of actions to be repeated as long as some condition holds.

The separation between procedural actions and non-procedural actions is an ill-defined one. In the early days of higher level languages such as FORTRAN, an assignment statement such as

```
X = X + A * B/C
```

was considered non-procedural since the programmer does not have to program the machine level instructions required to replace the variable X by a new value computed by executing the above statement once. On this level of the computational assignment statement, non-procedurality rapidly went as far as possible. However for the action statements which control the movement of data between levels of memory, the trend is towards more complex higher level statements. For the purpose of this section of the report, procedural language actions are identified as those which affect the access to, or a movement of, levels of data on or below that of an entry. Statements to open and close files are an exception to this, and are regarded as procedural. Statements which cause the processing of, or a set of accesses to, a whole file are regarded as non-procedural and will normally subsume the actions of file open and close.

GIS

No explicit statements to open and close files are provided nor to initiate specific transfers of data between levels of memory. However statements are provided to build up temporary files and to store data in temporary data areas in high speed memory.

MARK IV

No explicit statements are provided to transfer or move data across levels of memory or within a level of memory. Temporary variables in high speed memory may be defined. Arithmetic functions may be specified.

NIPS/FFS

RASP and the Output Processor, provide for temporary storage. Computational statements, subroutines and functions supply arithmetic capability.

TDMS

No explicit statements are provided to transfer or move data across levels of memory or within a level of memory. Temporary variables in high speed memory may be defined. Arithmetic functions may be specified.

UL/1

No explicit statements are provided to transfer or move data across levels of memory or within a level of memory. Temporary variables in high speed memory may be defined. Arithmetic functions may be specified.

COBOL

COBOL has an extensive set of action statements for all types of data transfer and for moving data within high speed memory. Arithmetic expressions may be embedded in COMPUTE statements to facilitate computation.

SC-1

No explicit file processing statements are provided. Statements may transfer data to temporary storage areas in memory and arithmetic computations are permitted.

4.1.2 Non-procedural language actions

The action statements provided in the non-procedural language of the self-contained systems are different from those embedded in host language systems. The action statements of the procedural language (see 4.1.1) are usually not provided, except for the arithmetic computation statements.

Since statements to control data transfer between levels of memory and to control movement within high speed memory are not provided, such actions are normally built into the self-contained system in the form of an implicit processing algorithm which takes over user control of data movement.

The two major processing algorithms are for interrogation described in this chapter and update (see Chapter 5). In both of these, the concepts of premise and action hold, although the premise is usually similar or even identical in both cases. Action statements in the interrogation function are directly concerned with generating and formatting reports for output. Action statements in the update function control the changes in value content to be made at some level in the data base.

The interrogation function is considered relevant only to self-contained systems and to host language systems which embody some capability for non-procedural interrogation or non-procedural report generation.

GIS

Action statements are LIST and HOLD which are normally used in a LOCATE - EXHAUST block. This block causes a looping through a set of entries or groups. These may be either actions such as LIST and HOLD, or else conditional statements. LIST causes the extraction of the item values whose names are included in LIST statement. HOLD similarly causes the build up of a temporary file.

MARK IV

The action statements are implied by the particular form being used. When the data on the form is punched, certain columns contain parameter values indicating actions to be performed.

NIPS/FFS

The actions in RASP are used to sort the file (or files) of selected entries. Generating reports must then be done using the separate OP facility, which gives the user control over report content (including possible further selection) and also over format. QUIP actions include statements for listing, printing and displaying of selected data depending on output medium.

TDMS

In system formatted reports, the PRINT statement with items names causes data to be included in the report. In the user formatted reports, which use COMPOSE, lower level actions to permit formatting the report are provided.

UL/1

The actions in the interrogation division are always prefaced by

```
            ⎧ REPORT  reportname         ⎫
PUBLISH     ⎨                            ⎬
            ⎩ [COBOL] FILE filename      ⎭
```

and there may be several similar PUBLISH specifications for any one selection criterion. In either case, this is followed by format specifications for the report or file.

COBOL

COBOL has no non-procedural language actions in the sense of this report. All looping must be programmed and actions on a file or set of files built up as a set of actions on the individual entries. However, the sort verb and the report writer facility are optional non-procedural actions available to the programmer from within the language.

SC-1

Action statements are used within a Report Description to call other Report Descriptions and a PRINT command is used on a control card at execution time to specify the main (calling) Report Description to be used. PRINT statements within each Report Description specify which items appear in the output report.

4.2 Identifying applicability of interrogation

The first specification for the interrogation function defines the level of applicability of a single interrogation, for instance whether it may operate on a single named file, a set of files, a single partition of one file or on a whole data base. If it operates on a set of files, the set may be coordinated, that is similarly sequenced. Otherwise the set is uncoordinated. In either case the size of the set may vary.

GIS

The set of files to be interrogated is listed in the QUERY statement which identifies the start of a QUERY sub-procedure. Up to 16 files may be interrogated in one specification. They need not be similarly sequenced.

MARK IV

The name of the file or files to be interrogated is entered on the Run Control form which is used in conjunction with both the Information Request form and the Processing and Record Selection form. The latter form must be used if a set of coordinated files is being interrogated. In this case the master and the associated files are specified on the run control forms. Using the facility to interrogate a set of coordinated files, there must be one master file and may be up to nine slaves.

NIPS/FFS

With all three interrogation facilities the file or files to be processed may be identified by a FILE statement. In the terminal facility QUIP, the format of the FILE statement provides for identification of the batch query used to extract the data and the report format to be used to print it. In QUIP, the QUERY statement may also be used to identify the file from which data is to be extracted.

In both RASP and QUIP, once the file is identified, different formats of the LIMIT statement may be used to confine the data extraction to stated ranges of entry identifier values. In QUIP, the FIND statement can be used to state the maximum number of entries to be extracted. In RASP, the SELECT statement can be used to reduce the size of the entries extracted by specifying only specific repeating groups or instances of repeating groups that will be included in the extracted entries. If SELECT is not used the entire entry is extracted and the instances of repeating groups that satisfy the entry level conditional expression are flagged.

TDMS

The name of the file to be interrogated is entered in response to the system request

ENTER DATA BASE FILE ID:

which precedes the specification of the interrogations. Only one file may be interrogated in one interrogation specification.

UL/1

The name of the file to be interrogated is included in the statement.

INTERROGATE filename

which precedes the specification of the interrogation. If a set of coordinated files is to be interrogated, then they may have one entry in the file definition data catalogue which identifies them as a set. The set must have one master file and up to three slave files.

COBOL

The set of files to be interrogated is identified by the OPEN and READ statements which are used in programming the file processing algorithm. There is no limit on the number of files which may be processed.

SC-1

The name of the file to be interrogated by a Report Description is identified in the first statement of the Report Description. One Report Description may call other Report Descriptions which interrogate other files.

4.3 Conditional expressions

Selection facilities typically permit a complex condition to be expressed on each instance of an entry, or a group. The condition on the entry or group consists of a logically connected set of conditions on one or more data items (usually but not necessarily on the item value).

GIS

A conditional expression is introduced either by IF or by WHEN. One may also be used after the extension word ON to provide selective report formatting when preparing a user specified report.

MARK IV

The form identification for the Information Request form or for the Processing and Record Selection form identifies the input being specified as an interrogation and in both forms the condition expression is built up in specific columns.

NIPS/FFS

A conditional expression is introduced by the word IF in all three interrogation facilities. There are variations in the allowable parts of the expression among the three facilities. In RASP, FURTHER can be used as a replacement for IF or as a connector in an conditional expression with an implied meaning of AND, if no other connector is used explicitly. In OP the IF statement may be preceded with the action verbs OMIT or STOP.

TDMS

A conditional expression is always introduced by the word WHERE. Conditional expressions are used in both QUERY and in COMPOSE and the same capabilities are provided in each case.

UL/1

A conditional expression is specified using the criterion language. As the selection criterion on the entry in the Interrogation Division, it is introduced by the words RETRIEVAL CRITERION. Within a computational procedure specified using the procedure language (see 4.7.4), conditional expressions are preceded by the word IF.

COBOL

A conditional expression is introduced by the word IF.

SC-1

Conditional expressions are preceded by IF for expressions in which numerical quantities are being compared and ALPHA IF for expressions in which alphanumeric quantities are being compared. Only simple conditions are permitted in a conditional expression.

4.3.1 Simple conditions

The smallest indivisible condition is referred to as a simple condition, which may take various forms depending on the system. The basic form is a subject, such as an item name, followed by a predicate. The predicate usually contains two parts, a relational operator, and a reference quantity. The common case of a simple condition containing subject, relational operator and reference quantity is called a relational condition. Some systems allow the subject or the reference quantity to have several values, but this is a shorthand way of writing several simple conditions as a single compound condition. Compound conditions are formed by connecting together two or more simple conditions (see 4.3.2).

4.3.1.1 Basic relational operators

The basic six relational operators are equals, not equals, greater than, less than, greater than or equal to, less than or equal to.

OPERATOR SYSTEM	EQUALS	NOT EQUALS	GREATER THAN	LESS THAN	≥	≤
GIS	EQ	NE	GT	LT	GE	LE
MARK IV	EQ	NE	GT	LT	GE	LE
NIPS/FFS[1] ALL options	EQ EQUAL(S) EQUALING	NE	GT	LT	GE	LE
RASP and OP batch			AFTER LATER	BEFORE EARLIER	GTE	LTE
OP batch		NEQ	NGT	NLT	NLTE	NGTE
QUIP	=		>	<		
TDMS	EQ	NQ	GR	LS	GQ	LQ
UL/1	EQ	NE	GT	LT	GE	LE
COBOL	= EQUAL(S)	NOT = NOT EQUAL UNEQUAL	GREATER EXCEEDS	LESS		
SC-1	EQ =	NE	GT >	LT <	GE	LE

[1] In all NIPS/FFS options, the six basic relational operators may be preceded by NOT to reverse the meaning.

Figure 4-1
Basic relational operators

4.3.1.2 Permitted form of subject in simple relational conditions

The subject in a relational condition usually identifies a data item in the data base schema. Most frequently it is an item name, although there are other possibilities such as the name of a quantity computed from data in the entry or some other property of the item named. The subject in the written form of a relational criterion is usually written to the left of the relational operator.

GIS

The subject is restricted to the item name. Which instance of the item name is selected, is determined by the place of relational conditions within the nest of LOCATE-EXHAUST blocks necessary to define the processing algorithm.

MARK IV

The subject may be an item name in the entry of the master file or possibly in any of its coordinated files (up to nine) or in the transaction file. In addition it may be a temporary work item or one of the systems flags used to allow the user some control over the processing algorithm. If the file is being updated and an entry modified, then the subject may also be an item name in the new file.

NIPS/FFS

All three facilities may use an item name as the subject of an IF statement. The item name may be qualified with the name of a conversion table or routine. This causes either a table look up or a subroutine call to produce the value which corresponds to the reference quantity to be used in evaluating the condition (see 4.3.1.4). In addition, RASP and OP can both use simple group names, sub-items and functions. OP may use a RASP generated value passed to OP in a temporary work item. In addition RASP may use a literal.

TDMS

The subject is restricted to an item name in the entry of the file being interrogated. As throughout the system, the item number may be used interchangeably with item name.

UL/1

The subject may be an item name in any entry type in the master file or possibly in any of the three coordinated files. In addition it may be the length of the stored value, or the picture of the item. If the item is numeric and multiple valued, the subject may be the arithmetic sum or mean of the set of values. If the item is multiple valued of any type, the subject may be the number of values, or the maximum or minimum of the set of values. In addition the subject may be the name of a computational procedure based on the values of numeric data items in the entry.

COBOL

The subject may be an identifier literal or arithmetic expression. The term identifier here is used in the COBOL sense and refers to data item names followed if necessary by the syntactically correct combination of qualifiers, subscripts, and indices necessary to make unique reference to a data item.

SC-1

The subject may be the name of an item of the file being interrogated, or the name of a temporary data item. The name of a temporary storage area containing the value of an item from another file may also be the subject of a conditional expression.

4.3.1.3 Restrictions on the use of relational operators

Sometimes it is meaningful to impose restrictions on the relational operators (see 4.3.1) permitted with a given subject. These may depend on the type of the item identified in the subject or in the form of the subject.

GIS, MARK IV, NIPS/FFS, TDMS, COBOL, SC-1

All permitted subject forms may be used with each of the six basic relational operators.

UL/1

All permitted subject forms may be used with each of the six basic relational operators, except PICTURE which is restricted to use with EQ and NE.

4.3.1.4 Permitted reference quantities in simple relational conditions

In a simple relational condition the subject is compared, on the basis of the relational operator, with a reference quantity. This is normally a literal agreeing in type with the subject but it may be another item identifier or an arithmetic expression.

GIS

The reference quantity must be a string value or string expression for subjects of type string, or it must be a numeric value or an arithmetic expression for subjects of type numeric. A string expression must be either a string value or a string variable. Two or more string expressions may be concatenated.

MARK IV

The reference quantity may be an item name, a temporary work item name, or a system flag. In addition it may be a string value or a numeric value and must then agree in type with the subject. String values are automatically converted to numerics if necessary. Numeric item of various types are converted to the form of the subject.

NIPS/FFS

The form of the reference quantity varies with which of the three facilities is being used. The widest range of reference quantity types is found in RASP, where data item names, simple group names, literals, indirect address literals, (i.e., literals that have been stored and given a name), and functions may be used.

In the Output Processor the reference quantity may be an item name, a simple group name, or a literal. The data item names must be from the fixed part of the entry or from the same repeating group as the subject.

In QUIP, the reference quantity may be an item name, a simple group name, a temporary item name or one or more literals.

TDMS

The reference quantity may be a string value or a numeric value depending on the type of the subject. If the reference quantity is one of the set of values which the item is known to take in the file, then the numeric code which the system assigns to the value may also be used in place of the reference quantity.

UL/1

If the subject is an item name, the reference quantity must be a value agreeing in type with the subject. It may also be another item of the same type. If the subject is the name of a numeric item, then the reference quantity may be a procedure name or a reference name (this being a numeric quantity computed in a procedure). Also, if the item is of type numeric, the reference quantity may be an item name for any type of item followed by the word REPEATS which references the number of values in the item instance.

If the subject is a procedure name, then the reference quantity must be a numeric value or another procedure name. If the subject is PICTURE then the reference quantity must be a valid picture. If the subject is a count on the number of values in a multiple valued item, then the reference quantity is an integer.

COBOL

The reference quantity (in COBOL called an object of the conditional) may not be a literal if the subject is a literal. Otherwise there are no restrictions. If the subject is numeric and the object non-numeric (or vice versa), then the numeric quantity is treated as though it were moved to an alphanumeric data item of the same size as the numeric quantity.

SC-1

The reference quantity may be an item name, a temporary data item name, a string value or a numeric value.

4.3.1.5 String matching conditions

A special kind of relational condition is that peculiar to alphanumeric or string data. A data item in the file is scanned to see if it contains the literal string expressed in the condition. A specific capability in such string matching is to allow one or more characters in the literal string to be "don't care" characters, which means that the data items may have either any, or any non-blank character in that position.

GIS

This is called a scan condition and uses the relational operator SN. The reference quantity may contain one or more instances of a special character (the lozenge) to indicate a "don't care" character and another special character to restrict the search to the beginning or end of the value indicated in the subject.

MARK IV

String matching may be performed at the beginning or end of the string.

NIPS/FFS

In a relational condition (as described in 4.3.1.1 to 4.3.1.4), it is permitted to specify that only the first and/or last few characters are to be used in evaluating the condition. Alternatively it is possible to indicate that these characters are to be taken as a universal match and the conditions evaluated on the remaining characters.

TDMS

No capability is provided.

UL/1

This capability is provided using the relational operators CONTAIN(S) and DO(ES) NOT CONTAIN. The reference quantity may contain one or more instances of a special character (the lozenge) to indicate a "don't care" character. By using string delimiters around the reference quantity, leading, terminal and/or embedded spaces may be included.

COBOL

This capability is provided by the INSPECT TALLYING statement.

SC-1

In a conditional expression the evaluation of the condition may be restricted to the leftmost characters of the subject only.

4.3.1.6 Other relational operators

Several systems allow relational operators other than the basic six (see 4.3.1), and those needed for string matching (see 4.3.1.5). Typically these are included to handle some special item type. Some require two or more reference quantities.

GIS

The relational operator BT, meaning between, followed by two reference quantities, is allowed. The condition

itemname BT A, B

is equivalent to the compound condition (see 4.3.2.4)

itemname GT A AND itemname LT B

NIPS/FFS

The extraction facilities in the Output Processor have no special relational operators. RASP and QUIP have three special relational operators. BETWEEN or BT is followed by a pair of reference quantities. The condition is true if the value of the subject is equal to or between the pair of reference quantities. RASP will test for multiple ranges in a single IF statement. QUIP will handle only one reference pair per statement.

RASP and QUIP also have two geographic relational operators, CIRCLE or CIR, and OVERLAP or OVP or OVERLAPS. Both of these operators require geographic coordinate data and will not accept any use of a universal match character, or conversion subroutines. The CIRCLE operator is used to determine whether the subject is with a circle defined by the geographic coordinates of a point and a distance. These two values are the reference quantities for this operator. The OVERLAP operator is used to determine if polygons bounded by from 1 to 8 points overlap. For lines, which are polygons defined by two points, intersection is interpreted as overlap.

TDMS

A range expression may be used as the object of a relational condition using the relational operator EQ. The range expression consists of a lower bound and an upper bound separated by three dots.

SALARY EQ 500... 800

MARK IV, UL/1, SC-1

No other relational operator is provided.

COBOL

COBOL allows two other kinds of condition called class conditions and sign condition. A class condition may be used to test whether a value consists entirely of numeric characters. It may also be used to test whether a value consists entirely of alphabetic characters. These conditions may be used only on alphanumeric items (in COBOL called display items).

A sign condition is used to determine whether or not the value of a numeric data item or of an arithmetic expression is less than, greater than or equal to zero.

4.3.1.7 Existence conditions

An existence condition consists of a subject and a verb. The subject has the same role as in a relational condition. The verb takes the place of the relational operator but is not normally followed by a reference quantity. The existence condition checks the presence or absence of a value in an instance of an item.

GIS

An existence condition is expressed in one of two forms

itemname AB

where AB means absent, is true if a value has not been supplied for the item.

itemname EM

where EM means empty, is true if the item is packed decimal or right-justified EBCDIC and its value is zero. It is also true if the item is left-justified EBCDIC and its value is all spaces.

MARK IV

An existence condition on a fixed or variable length item may be expressed by a relational condition using EQ and a value such as zero or all spaces as the reference quantity. This test is invalid if either of these are legal values. If the item is variable length, then the condition may also be written

itemname EQ itemname

where the two itemnames are identical. If the value is missing the test fails.

NIPS/FFS

An existence condition on an item may be expressed by a relational condition using EQ and a value outside the legal range of values for the item, or all spaces as the reference quantity.

TDMS

An existence condition may be expressed in one of two forms

itemname EXISTS
itemname FAILS

The first is true if a value has been supplied for an item, the second is true if no value has been supplied.

UL/1

An existence condition may be expressed in one of two forms

itemname EXISTS

itemname DO(ES) NOT EXIST

The first is true if a value has been supplied for an item, the second is true if no value has been supplied.

COBOL

An existence condition must be expressed by a relational condition using EQUALS and a reference quantity such as zero or all spaces to designate an absent value.

SC-1

An existence condition may be specified by comparing a numeric item to zero or an alphanumeric item to blank.

4.3.2 Compound conditions

Compound conditions are built up from simple conditions using logical connectors. Normally some precedence rule for the connectors is applied. In specifying a compound condition some degree of implicitness may be allowed to minimize the time required to write lengthy compound conditions containing a degree of repetition.

GIS, MARK IV, NIPS/FFS, TDMS, UL/1, COBOL

All systems allow the specification of compound conditions.

SC-1

Compound conditions may be programmed by nesting conditional expressions or using control transfers. Explicit compound conditions are not provided.

4.3.2.1 Logical connectors

The most common logical connectors used are AND and OR. Other possibilities are NOT, NAND and NOR the last two of which can be synthesized from AND and OR. If the appropriate relational operators are provided, it is also possible to synthesize situations where NOT would be used. The form of expression for the connector also varies from system to system. In free form languages, the form AND, OR, NOT is usually preferred although in some systems a single character such as & or * may be required. In a forms oriented system, the logical connector may be indicated by a suitable character in the appropriate column.

CONNECTOR / SYSTEM	AND	OR	NOT	NOR	NAND
GIS	AND &	OR ,	NOT		
MARK IV	A	O			
NIPS/FFS					
All extraction options	AND	OR			
RASP batch retrieval		,			
Terminal retrieval	&	|	NOT		
TDMS	AND	OR	NOT		
UL/1	AND	OR			
COBOL	AND	OR	NOT		
SC-1	Not applicable				

Figure 4-2
Form of logical connectors

4.3.2.2 Precedence rule for logical connectors

There are four possible precedence rules for the expression

A AND B OR C

These are

- AND takes precedence
- OR takes precedence
- expression undefined
- left to right precedence

If AND takes precedence, the expression is true if both A and B are true or if C is true. If OR takes precedence, it is true if A is true and if either B or C is true, or both B and C are true.

SYSTEM	PRECEDENCE
GIS	AND
MARK IV	AND
NIPS/FFS[1]	AND
TDMS	AND
UL/1	none
COBOL	AND
SC-1	n.a.

[1] All three facilities operate this way. If the Output Processor is used, however, this must be forced by putting a BOOL parameter in the CREATE card that initiates batch print out jobs. Without the BOOL parameter OR takes precedence.

Figure 4-3
Precedence rules

4.3.2.3 Levels of nesting

When several conditions are grouped together, it is usually possible to nest conditions logically to permit the expression of very complex selection criteria. Typically, in a free form language, parentheses are used to nest conditions and hence the number of left parentheses to the left of a condition is a possible count of the level of nesting. In a forms oriented system, a level count may be entered in one column in the form.

In systems with no limit on the number of parenthetical levels, there are usually other system limits - such as high speed memory available when the conditions are translated, maximum number of characters allowed per statement, or maximum number of item and/or group names per data extraction, which normally act as limits on condition nesting.

The number of levels permitted in each system is:

GIS	4
MARK IV	9
NIPS/FFS	
OP	0
RASP	8
QUIP	no limit
TDMS	no limit
UL/1	no limit
COBOL	no limit
SC-1	not applicable

4.3.2.4 Compound conditions on the same item

A set of conditions on the same or on different items is used to build up the total entry level conditional expression. How successive simple conditions must be connected differs in the degree of repetition of the subject and the relational operator when these are the same in successive conditions. The simplest way to connect two or more simple conditions is when both have the same subject and the same relational operator with differing reference quantities.

GIS

Two or more simple relational conditions on the same item may be expressed without repeating the item name, as long as the relational operator is the same. In such cases the comma is used as a separator to indicate the logical connector OR and the ampersand to indicate the logical connector AND.

```
SKILL  EQ  'PLUMBER' , 'CARPENTER'
SKILL  NE  'PLUMBER' & 'CARPENTER'
```

If the relational operator changes, the item name must be repeated.

```
SALARY  EQ  500  OR  SALARY  GE  800
```

MARK IV

Two or more simple relational conditions on the same item are written on successive lines on the same form. One column is used to enter a code, A or O, indicating the logical connector (AND or OR) by which the condition on that line is connected with that on the previous lines. The item name must be repeated even if it is the same as on the preceding line.

NIPS/FFS

Two or more relational conditions on the same item may be expressed without repeating the item name as long as the relational operator is the same. A comma or space is used as a separator between successive reference quantities.

```
SKILL     EQ   PLUMBER,    CARPENTER.
SKILL     NE   PLUMBER,    CARPENTER.
SALARY BETWEEN 100/300, 500/800.
```

When the relational operator is negative, i.e., NE, the comma or space has an AND effect, otherwise it has an OR effect.

If the relational operator changes, the item name must be repeated.

```
SALARY  GE  500  AND  SALARY  LE  800
```

TDMS

Two or more relational conditions on the same item may be expressed without repeating the item name for the relational operators EQ and NQ. The comma is used as a separator between successive reference quantities and its interpretation depends on the relational operator used.

```
SKILL  EQ  PLUMBER, OR CARPENTER
SKILL  NQ  PLUMBER, AND CARPENTER
```

UL/1

Two or more relational conditions on the same item may be expressed without repeating the subject or relational operator.

```
SKILL  EQ  PLUMBER  OR CARPENTER
SKILL  NE  PLUMBER AND CARPENTER
```

If the relational operator changes, it is not necessary to repeat the subject.

```
SALARY  GE  500   AND   LE   800
```

COBOL

Two or more relational conditions on the same item may be expressed only when multiple reference quantities occur. In such cases, the subject and the relational operator may be omitted from the second relational condition.

```
SKILL     = PLUMBER OR CARPENTER
SKILL NOT = PLUMBER AND CARPENTER
```

SC-1

No capability is provided.

4.3.2.5 Logically connected conditions on different items

There are various conventions for connecting conditions on different items. In some systems the only connector may be AND and this is implicit. In most cases both AND and OR may be used with the precedence rule (see 4.3.2.2) being invoked.

GIS

Conditions on different items may be connected by AND or OR. In either case the connector may be followed by NOT to negate the one condition following. Each condition is a separate statement and must therefore be entered on a separate line.

```
SKILL   EQ   PLUMBER
AND NOT SALARY GT 500
```

MARK IV

Two or more relational conditions on different items are written on successive lines of the same form. One column is used to enter A or O indicating the logical connector by which the condition on that line is connected with that on the previous line.

NIPS/FFS

Conditions on different items may be connected by AND or OR.

```
SKILL EQ PLUMBER AND SALARY GT 500
```

TDMS

Conditions on different items may be connected by AND or OR.

```
SKILL EQ PLUMBER AND SALARY GR 500
```

UL/1

Conditions on different items must be written successively without the connector if it is AND

SKILL EQ PLUMBER SALARY GT 500

If the connector is OR, each condition must be assigned a name and the expression connecting the two conditions expressed at the end of all items conditions in an expression called the multilevel criterion.

COBOL

Conditions on different items may be connected by AND or OR.

SKILL = PLUMBER AND SALARY NOT > 500

SC-1

No capability is provided.

4.3.2.6 Logically connected conditions on several items but with the same reference quantity

In some systems there is a special capability allowing an abbreviated form for expressing two or more relational conditions which have different subjects but the same reference quantity. This would be an expression of the form

HIREDATE OR RAISDATE EQ 701201

TDMS

Items of the same type may be used as a list with AND or OR preceding final member of the list. There must be one relational operator and one reference quantity.

BIRTHDATE, HIREDATE, OR RAISDATE EQ 12/1/70
HIREDATE, AND RAISDATE LS 12/1/70

GIS, MARK IV, NIPS/FFS, UL/1, COBOL, SC-1

No capability is provided.

4.4 Conditions on groups

If the entry schema class allows for groups (see 2.2), specifically for repeating groups and group relations, then capability may or may not be provided to place conditions on groups. This specifically excludes placing conditions on the principal items in entries containing group structure. Conditions on principal items may be expressed using the basic facilities for conditional expressions (see 4.3). When facility is provided for representing groups then it is sometimes necessary to place a condition on the items in the group as if the group structure did not exist.

The placing of conditions on groups, taking into account the group structure, may be provided for one of two purposes. First of all the user may be seeking an entry instance (namely data from the principal items) for which the group structure satisfies some conditions. In the second case, the user may be looking for the data contained along some path through the group structure. In this case the system must not only evaluate whether the conditions exist, but must also retain a notation of which paths through the structure actually satisfy the conditions. A further refinement of the above comes in the requirement to check for the existence of, and possibly retain, two or more paths in an entry. In this case the separation of the extraction (see 4.6) from the conditional expression becomes less appropriate.

In summary, the following capabilities can be identified in connection with the expression of logical conditions on data in groups where the group structure is taken into account.

- check for existence of a set of logical conditions on different items in the group
- as above, but noting (in some sense) the data items which satisfy the conditions
- checking for the existence of two or more (including all) paths through the structure which satisfy a set of logical conditions
- as in the preceding case but noting the count of satisfactory paths
- as in the preceding case but noting also the data items (and hence the paths)

In the case of expressing logical conditions on groups, there are two basic approaches to facilitating this. The procedural capability may be adequate to allow the user to specify all such conditions by means of conventional programming practices, namely, looping through group and item instances, testing values, and storing hit values in temporary work items. Alternatively, a non-procedural approach may be provided in which special language elements imply a connection between conditions on different levels of the hierarchy.

GIS

The procedure control provided permits specification of all capabilities which take into account group structure. Looping through group and item instances within a structure is specified in LOCATE EXHAUST blocks. Numbers of hits can be accumulated in the special temporary items, called TALLYn and values along paths can be stored in the other temporary items, as long as the limit on these is not exceeded (see 4.).

MARK IV

No capability is provided on the Information Request form or the Processing and Record Selection form. However use of own code hooks to routines written in COBOL and FORTRAN would permit these kinds of conditional expressions to be written.

NIPS/FFS

In RASP the retrieval logic used with the IF statement, flags the instances of the repeating groups in the entry that meet the conditions of the IF statement. The user may also write a Function Operator subroutine. When this is done, the Function Operator can be used as one of the clauses of the IF statement. The subroutine is written in a procedural language, and all the capabilities listed may be programmed.

TDMS

Some non-procedural control is provided for matching a condition on one level in the structure with a condition on a lower level. The word HAS is used, as in the following example

```
WHERE SKILLCDE EQ 5520 AND
SKILLCDE HAS JOBTITLE EQ FILE CLERK
```

This facilitates checking for the existence of conditions with a nested group structure.

The system also provides facilities for extracting data from groups on any level in the structure. The words HAS EVERY may be used to specify that all subordinates for a given group instance satisfy a condition.

For example

```
JOBCODE EQ 1330 AND
PERSONS HAS EVERY EMPJOBCODE NQ 1330
```

could be used to select organizational units which had an authorized job code of 1330 but none of the employees in the unit actually had that code. It is not possible to count the number of paths through a structure and place a condition on that count.

UL/1

The facilities required to express conditions on nested structures is provided in the procedure language which is used principally to express computations on numeric data. For this reason only numeric data can be stored in temporary items. Counts can be accumulated within the procedure but paths containing data satisfying the conditional expression cannot be retained for use by the extraction facilities (see 4.6).

COBOL

All situations can be handled procedurally by use of loops, tests and control transfers.

SC-1

Embedded repeating groups may be processed by Report Descriptions which are called by the main Report Description. Conditional expressions may be used in the called report descriptions to test items within the repeating groups, and the results of the test may be passed back to the main report description through temporary items. All of the capabilities listed could be handled. The more complex tests would be programmed using several nested report descriptions and using temporary data items as switches within the logic of each Report Description.

4.5 Identification of conditional expression

If a conditional expression (or even part thereof) is to be used frequently, it is useful to assign a name to it and to cause the named expression to be stored in a file of similar expressions for subsequent invocation.

GIS

It is not possible to assign a name to a conditional expression. However any part of a sub-procedure may be named and cataloged for subsequent invocation.

MARK IV

It is not possible to assign a name to a conditional expression. However a whole interrogation may be given a request name and stored in a catalog. When it is invoked some modification of what has been stored can be accomplished by use of temporary items.

NIPS/FFS

Queries may be named and cataloged. They may be stored with parameters that must be supplied at execution time, with default parameters that will be used if none are supplied at execution time or as a complete self-contained unit.

TDMS

It is not possible to assign a name to a conditional expression. However, the last previously used expression may be referenced by use of the word SAME. A report description may be saved, together with dummy parameters, for later invocation. The parameters may be replaced with values at execution time.

UL/1

A conditional expression or any part of it may be defined as a macro. It is stored in a file and has no meaning after completion of the use of the interrogation function: it may, however, be invoked several times and may have several parameters. Default values may be assigned to these at definition time. At invocation time values must be assigned for parameters having no default values; over-riding values may be assigned to parameters having default values.

COBOL

It is not possible to assign a name to a conditional expression. However, a conditional expression can be embodied in a procedure and performed by other parts of the program.

SC-1

Conditional expressions cannot be named or stored. Report Descriptions containing conditional expressions can, however, be stored and retrieved.

4.6 Data extraction

Data satisfying the conditional expression comprising the premise may be extracted and placed either into report form or into machine readable form possibly after intermediate processing. For the purpose of analysis, the process of extraction is separated from that of formatting. Extraction considers which quantities on each structure level (item, group, entry, file) may be included in the report or output file, irrespective of the process of formatting. Separate consideration is given to the features provided for formatting printed reports (see 4.11) and for formatting files for further processing (see 4.13).

GIS

Data extraction may be specified in a QUERY subprocedure (see 4.) with report writing statements and with statements which cause writing to a hold file. Both types of statement may be in the same subprocedure. Data extraction may also be specified in other kinds of procedures, such as MODIFY or UPDATE (see Chapter 5).

MARK IV

Data extraction may be specified using either the Record Selection part of the Information Request form or the Output Content form, the latter being used in conjunction with the Processing and Record Selection forms.

NIPS/FFS

For extraction from one file in on-line mode, QUIP may be used only on direct access files, either sequential or index sequential. In the batch mode, either OP or QUIP may be used to publish data from a single file. RASP extracts data from more than one file, identifies it by QRT/QDFs, and places it in an answer file from which OP can be used to write reports.

TDMS

Data extraction may be specified using either COMPOSE or QUERY. COMPOSE is a facility for interactively preparing a potentially complex report format. QUERY is used only in an interactive mode and the report formats are fairly elementary.

UL/1

The action part of an interrogation is specified in the Publish Section which allows the user to output either reports or machine readable files. In either case, the format of the content and summary data may be either system supplied or user specified.

COBOL

Use of the Report Writer feature relieves the user of considerable Procedure Division programming. All formats must be specified by the user, who is however relieved of such actions as writing procedures for moving data, constructing print lines, counting lines on a page, numbering pages, producing heading and footing lines when required, recognizing the end of logical data sub-divisions and updating sum counters. These operations are all accomplished by the Report Writer Control System as a consequence of statements that may be included in the Report Section of the source program's Data Division.

The Report Writer statements to be used in the Procedure Division are INITIATE, GENERATE, TERMINATE, SUPPRESS and USE BEFORE REPORTING. The main loop of the report generation would normally contain READ and GENERATE statements.

SC-1

Data extraction is specified using report language statements within each Report Description. These statements process data items from an entry and print these data items, temporary work items, or literals in a report format specified by the user in the non-procedural section of the Report Description.

4.6.1 Generalized extraction features

Facilities may be provided for allowing the specification of reports with system control over format, or with user control over format. In addition it may be possible to extract files, again with either user or system control over format.

SYSTEM	REPORTS		FILES	
	USER FORMATTED	SYSTEM FORMATTED	USER FORMATTED	SYSTEM FORMATTED
GIS	yes	yes	no	yes
MARK IV	yes	yes	no	yes
NIPS/FFS	yes	yes	yes	yes
TDMS	yes	yes	no	no
UL/1	yes	yes	yes	yes
COBOL	yes	no	yes	no
SC-1	yes	no	no	no

4.6.2 Multiple outputs per conditional expression

The user may specify one report or file to be produced from the data entries satisfying the conditional expression. However, two or more outputs may be specified for the same conditional expression, without respecifying the expression. These multiple outputs may be extracted on to different output media.

GIS

In a QUERY sub-procedure, there is no fixed relation between any conditional expression and the action statements which cause data to be extracted. Therefore multiple outputs per conditional expression on an entry are possible. The outputs may include any mix of output files and reports.

MARK IV

Up to nine reports and/or up to nine extracted files may be generated for any one entry level conditional expression.

NIPS/FFS

The number of extractions per entry level conditional expression depends on whether use is made of previously defined and stored report formats or not. It so there is no limit; if not there is a limit of 40 reports per run.

TDMS

Only one report may be generated for each entry level conditional expression.

UL/1

Any desired mix of output files and reports may be specified, any or all of which may require sorting. The limit on the number of outputs allowed is governed by the amount of core assigned to a given run within an overriding limit of 43.

COBOL

Each RD entry in the Report Section of the Data Division forms the complete description of one report. It must be associated with a file in the File Section and the file is assigned to an output device in the normal manner. More than one report may be associated with the same file.

SC-1

Only one report may be generated for each PRINT command. Entry level conditional expressions may qualify the data extracted and used for this report.

4.6.3 Limit of output volume

The facility of limiting the output volume from an interrogation set may be provided. Examples are an upper limit of the number of entries satisfying the conditional expression, or an upper limit on the number of report pages to be printed.

GIS

This facility may be achieved by using a temporary work area to store a count of the number of entries satisfying the conditional expression, and jumping out of the programmed loop when this number reaches a certain pre-specified value.

MARK IV

Two portions of the Run Control form, called Start Search and End Search are used to limit the volume of output. In each of these 16 byte areas, the user may write a value of the entry identifier. Searching will automatically bypass entries with a value lower than that in the Start Search area and with a value higher than that in the End Search area.

On the Processing and Record Selection form and on the Information Request form, a box called "maximum items selected" (the term "items" here implies "entries") may be used to limit the number of hit entries. Also it is possible to limit output from hierarchical structures when multiple hits are possible.

On the Output Format Specification form, the user may enter a number between 1 and 9999 to limit the maximum number of pages for the report.

NIPS/FFS

File searches of a specific file may be limited to specified ranges of values of the high order positions of the entry identifier items. If only one file is being searched, a primary limit may be further refined by individual retrievals within the batch. If a retrieval is being made from more than one file, the retrieval must specify completely its own entry identification limit for each file.

Extraction of the file data may be conditioned on whether it satisfies some function of file data values and execution time specified parameters. Production of output reports may also be conditioned on file data values or parameters such as number of records processed or number of pages produced through use of a conditional STOP statement.

TDMS

No facilities are provided in COMPOSE for limiting the volume of output. In QUERY, a LINES statement may be used to limit the number of lines output by a PRINT statement. The user is notified when the limit is reached, and he may terminate the printing or request a continued output of the same number of lines.

UL/1

No facilities are provided for limiting the volume of output.

COBOL

Output of a report can be terminated by tests on any of the fixed data names, LINE-COUNTER or PAGE-COUNTER, followed by a use of the TERMINATE statement to perform all of the functions associated with the termination of a report. A counter in the READ loop with tests on data in each entry could also be used.

SC-1

An upper limit on the number of entries processed by the report generator may be used. No upper limit on the number of pages may be specified.

4.6.4 Sorting

Sorting extracted data is a fairly common feature of the interrogation function. The sort may be provided by the operating system or there may be a special sort. Regardless of this, features of the sorting process include the specification of sortkeys, their names and order (hierarchy), the maximum number of characters which can comprise any sortkey, the maximum number of data items which can be components of a sortkey, the maximum number of characters which can be contained in all sortkeys, the types of sequencing provided for sorting entries (e.g., ascending descending), and whether the user has a choice of collating sequence.

The term "sortkey" is used here to indicate that an item which is designated as a sortkey is used to order the selected data for output purposes only. This must not be confused with an "identifier" or "sequencer" (see 2.3) both of which may play a role in the data structure.

GIS

To obtain a report in a sequence different from that of the file being interrogated, the user must write a QUERY sub-procedure to extract data into a hold file, a SORT sub-procedure to sort the hold file in the desired sequence and a second QUERY sub-procedure to produce the report from the hold file. A SORT sub-procedure uses the single statement

$$\text{SORT filename} \left\{\begin{matrix}\text{ASC}\\ \\ \text{DES}\end{matrix}\right\} \text{itemname1,} \left\{\begin{matrix}\text{ASC}\\ \\ \text{DES}\end{matrix}\right\} \text{itemname2,...}$$

Up to 64 items of any type may be used as sortkeys. ASC means "ascending", DES means "descending". The sortkey items must be from the root group of the entries in the hold file.

MARK IV

Using either the Information Request form or the Processing and Record Selection form, the user can designate up to nine items of any type as ascending or descending sortkeys.

NIPS/FFS

In RASP, one set of data extracted from a file may be sorted on up to ten different keys, up to ten different sets of extracted data may be sorted on one key each, or combinations of these not using more than ten keys may be sorted in a single batch run. In QUIP, only one key may be used at a time. Each key may be any number of data items as long as the total number of characters to be sorted does not exceed 210. The data items must be either items, simple groups, and/or items from one repeating group. In any of these cases the sort may be limited to either the high or low order positions of an item so long as the item is not a literal, binary number, or a geographic coordinate.

Output conversion routines may be applied to sortkey items before they are sorted. The order in which the items are sorted is determined by the order in which they are written in the SORT statement. All sorts used the EBCDIC collating sequence. RASP uses the OS/360 sort and QUIP uses its own.

TDMS

No sorting is possible using the inter-active QUERY facility. Using COMPOSE, up to ten items may be specified as either ascending or descending sortkeys. A special collating sequence of space, numbers, period, letters and special characters is used. No actual sorting of data takes place at execution but the ordering of the output report is faciliated by use of the indexes on each item. The form of the sort statement is

$$\text{SORT BY} \left\{\begin{matrix}\text{HIGH}\\ \\ \text{LOW}\end{matrix}\right\} \text{itemname1,} \left\{\begin{matrix}\text{HIGH}\\ \\ \text{LOW}\end{matrix}\right\} \text{itemname2...}$$

LOW is the default order.

UL/1

Up to 15 ascending sortkeys may be specified for any one report or file. The sortkeys may be any item, item length, date part or numeric quantity derived from items in the entry. The sorting facility is a specially built tag sort using disc working space and optional tape overflow. The EBCDIC collating sequence is used.

COBOL

The SORT verb may not be used within the READ and GENERATE loop. A READ loop may be used to pass entries to the sort. After the last entry has been passed the sorting is performed. The entries may then be returned to the executing object program one at a time via a RETURN statement. After each execution of this, a GENERATE statement may be used to report the sorted data.

SC-1

To obtain a report in a sequence differing from the sequence of the data base, the user must provide the definition of the structure of an extracted file including definitions of the desired data items and the desired sequence. He then invokes a utility program to populate the extracted file. The extracted file is then interrogated to produce the sorted report. Any number of sortkeys may be used; the maximum number of characters in all the keys cannot exceed 255. Ascending or descending order may be specified for each key; no choice of collating sequence is provided.

4.7 Item level extraction

Reports may be divided into two categories which are not necessarily mutually exclusive. The first category is that which contains item values or other item level attributes from the file. The second category contains only summary data based on a processing of data in the file, but no actual item level values from the file (see 4.10.2).

It may be possible to include in a report not only item values, but also item schema attributes such as items (see 2.1.2) and item value class attributes (see 2.1.2.2) .

4.7.1 Item schema attributes

Item schema attributes such as item name and type may be specified for inclusion in a report.

GIS

The column heading, which may be specified for each item at data definition time, is automatically included in system formatted reports generated using the LIST statement.

MARK IV

The column heading, which may be specified for each item at data definition time, is automatically included in systems formatted reports.

NIPS/FFS

If the QUIP user does not specify an output title or label for an item at data definition time (see Chapter 3), the item name is appended when an item value is reported.

TDMS

To include the item name in a report the name must be entered when the report is specified.

UL/1

Using the system formatted report facility, item names or item descriptions specified at data definition time are automatically included in both formats produced. (The format used and the choice between item name and item description depend on availability of space.) In the user formatted report facility, the item name and/or the item description may be specified for inclusion in the report. A code list for any coded item may be generated at the beginning of a report.

COBOL

Since there is no stored data definition available at execution time, no item schema attributes may be included in a report.

SC-1

Item schema atrributes are available in a report description only if they are included as literals in the report description.

4.7.2 Item instance attributes

The value of an item may always be included in a report. Other examples which may be included are existence status, date stamp and length of value.

GIS, MARK IV, NIPS, COBOL, SC-1

Value is the only attribute of an item which may be included.

TDMS

The maximum number of characters the set of values for an item is found to have may be displayed in a special report using the DESCRIBE statement.

UL/1

The attributes which may be included in a report are value, length of value, and date stamp, if the item schema indicates that one is used.

4.7.3 Item value decoding

If the set of values which may be assigned to item instances is predefinably finite then it is often allowed to define an associated value representation for each value in the set. The associated value may be longer than that stored in the entries and in this case, the latter could be referred to as the encoded form, and the other as the decoded form, of the value.

On extraction of an item it may be permissable to include either form in the report. In addition it may be possible to include at the beginning or end of the report a table giving the two forms for each member of the value set.

GIS

Any decoding defined for an item at data definition time is automatically performed during item value extraction. Decoding for all items may be overridden on any given interrogation through the use of the IGNORE statement.

MARK IV

The table look-up facility allows an associated value to be included in a report instead of the one in the file.

NIPS/FFS

During processing of the file of entries satisfying the conditional expression, the Output Processor and QUIP automatically call either a functional subroutine or a table look-up processor to decode values. They will call these if the subroutine or table was specified during data definition. The user may suppress decoding called for by data definition, or may supply his own subroutine or table to decode any item or non-repeating group.

TDMS

In COMPOSE, there is a capability to define and invoke a table look-up conversion for any item to be output.

UL/1

Items of type coded may be decoded for inclusion in a report.

COBOL

No capability is provided.

SC-1

The table look-up facility allows an associated value to be included in a report instead of the one in the file.

4.7.4 Item level derived data

Items of type numeric may be subject to computation to produce associated derived values for inclusion in the report. In some cases only limited arithmetic computation may be performed on individual items.

The capability to specify the inclusion of computed data may be identical for each level in the data structure in which case the separation of the computation of derived data into item level, group level (see 4.8.1), entry level (see 4.9.1) and file level (see 4.10.1) may be arbitrary. The capability for specifying computation is usually tied in with the file processing algorithm and also with the statements used to format the computed quantities in report of output file. The tie-in with the file processing refers to the systems facility to handle temporary data items as the different entry instances are examined. The tie-in with printing is a more semantic one; namely, the same statement form is often used to indicate that a quantity shall be printed as is used to indicate that it should be computed.

GIS

An arithmetic expression containing one or more items names and also constants can be specified for inclusion in either a system formatted or user formatted report or in a hold file.

MARK IV

No specific item level capability is provided.

NIPS/FFS

An arithmetic expression using the operators ADD, SUB, MUL, and DIV may be specified. It may use item names and constants. The result of computations and the printing of results may be conditioned on the item values and on the report format.

TDMS

With the COMPOSE facility, the user may incorporate derived values in the statements CONTENT and RECAP. These may contain arithmetic expressions as well as item names or literal strings. Arithmetic operations including addition, subtraction, multiplication, division, exponential, square root, logarithm, integral part and absolute value may be used. In addition six trigonometric functions (including inverses) are permitted. The QUERY print statement may also include arithmetic expressions.

UL/1

For multiple valued numeric items, use of the words SUM, MEAN, MAXIMUM or MINIMUM causes computation of the appropriate function for the set of values in the item instance.

Further arithmetic computation on numeric data items may be specified using the Procedure Language. This allows the user to define a computation procedure either in the Establishment or Revision Division, in which cases it is stored with the data definition data, or at the beginning of the Interrogation Division. The Procedure Language allows the five operations addition, subtraction, multiplication, division and integer division.

COBOL

Use of ADD, SUBTRACT, MULTIPLY and DIVIDE and of the COMPUTE statement is allowed in conjunction with the Report Writer facilities. This statement allows addition, subtraction, multiplication and division operations to be specified.

SC-1

An arithmetic expression containing a numeric item name, constants, and temporary item names may be used. Arithmetic operations permitted are addition, subtraction, multiplication, and division.

4.8 Group level extraction

The ability to include a whole group in an extraction falls into two classes. First, it may be possible to include a simple group by a use of the group name rather than by an aggregation of item level specifications. Secondly it may be possible to select for extraction a specific instance of a repeating group within an entry instance (see 4.4). In the latter case the capability to perform computation on selected instances of a group may be provided.

GIS

It is not permitted to include the name of any kind of group in a LIST statement. Specific group instances may be located by testing of item values within a LOCATE/EXHAUST block and the items from the groups selected may be named individually for extraction.

MARK IV

It is not permitted to include the name of any kind of group in the Output Content Specification form. Selection of specific group instances within an entry instance is possible only by storing the individual items in temporary items.

NIPS/FFS

In RASP, a SELECT statement may be used to include all instances of a repeating group, or only those satisfying a prestated conditional expression. Alternatively a third option allows extraction of all instances if a prestated condition is satisfied. In each case the entry's principal items are also extracted.

TDMS

Supplying the group name implies obtaining all values of all items within the group. Specific group instances can be selected for inclusion in a report by use of a conditional expression.

UL/1

When specifying a system formatted report, it is permitted to include the name of either a naming group or of a repeating group. This is then treated as if all members of the group were named individually. It is not possible to select a specific group instance within an entry instance for inclusion in a report unless the items to be included are numeric, in which case the procedure language can be used.

COBOL

The standard MOVE statement is used to move data into a reserved area which is used by the report generation facilities to output the report using a GENERATE statement. Since the MOVE statement may operate on any level of naming group ("group item") it is possible in this way to include groups in a report. Selection of specific group instances may be performed procedurally using a combination of IF and MOVE statements within the READ/GENERATE loop.

If no manipulation of the data is to be specified, then the SOURCE statement can be used to access the data directly, since a SOURCE statement is an implicit move.

SC-1

There is no automatic facility for providing group names in a report. Instances of a repeating group within an entry may be selected by specifying that an item from the repeating group must equal a specified value. If no such specification is made all instances of the repeating group are processed. Selection of instances may be made within the report description which processes entries of the repeating group.

4.8.1 Group level derived data

The facility for deriving computed quantities based on numeric data contained in group instances may be identical to that provided for the item level (see 4.7.1) or for the entry level (see 4.9.1). Capability provided normally depends on the degree of procedural control and would normally be closely associated to the facilities for expressing logical conditions on groups (see 4.4).

GIS, MARK IV, NIPS/FFS, TDMS, UL/1, COBOL, SC-1

The capability provided is a combination of that for expressing logical conditions in groups (see 4.4) and that for entry level derived data (see 4.9.1).

4.9 Entry level extraction

It may be possible to include a complete entry in an extraction by use of a single statement.

GIS

A complete entry may be included in a system formatted report using the statement

LIST RECORD

which causes all items to be listed in order of occurrence in the entry. An analogous facility is available for including complete entries in hold files.

NIPS/FFS

Using RASP, the SELECT statement may be omitted (see 4.2). This causes the whole entry to be included in the extraction.

TDMS

A complete entry may be included in a systems formatted report using the statement

PRINT ENTRY

which causes all items to be listed in hierarchical order.

UL/1

A complete entry may be included in a systems formatted report or a systems formatted file using the statement

PUBLISH RECORD

which causes all items to be listed in ascending order of item number.

COBOL

The SOURCE statement may be used to include a whole record in a report.

MARK IV, SC-1

No capability is provided.

4.9.1 Entry level derived data

If the entry contains numeric items, there may be capability to derive and extract quantities based on the values of two or more numeric items in an entry instance. This capability may supplement or completely subsume that for providing item level derived data (see 4.7.4) which applies only to a single item. This capability normally subsumes that for group level derived data (see 4.8.1).

GIS

The facility for including arithmetic expressions in a QUERY sub-procedure allows the expression to include several item names as well as literals and the names of temporary working items.

MARK IV

To perform computation across items in an entry, temporary items must be defined using the Temporary Field Definition form. The Processing and Record selection form then allows the computation of new quantities. On each line of the form a three operand statement may be embedded which assigns the sum, difference, product or division of the first two operands to the third.

NIPS/FFS

Arithmetic expressions may be included which allow numeric items to be added to or subtracted from a temporary item.

TDMS

Computation on two or more items in an entry may be specified in COMPOSE using the statements CONTENT and RECAP (see 4.7.4).

UL/1

The Procedure Language may be used to compute quantities based on two or more items (see 4.7.4). Any procedure which operates on an intra-entry level is called a "record oriented procedure".

COBOL

See 4.7.4

SC-1

Arithmetic expressions which contain one or more numeric items may be used.

4.10 File level extraction

It is not normally possible to extract a single file from a data base using one statement. If multi-file capability is provided, then the file in the data base on which the interrogation is to be performed is normally identified explicitly in an initial part of the interrogation statement.

Given that a set of entries are being extracted, there are several kinds of capability associated with deriving reports from across the set. These may be identified in two classes, the first containing the frequency of occurrence of file values and the second containing values computed directly from those in a file by a process other than counting frequency of occurrence.

4.10.1 File level derived data

The computation of data based on that contained in several entries may be achieved by one of two basic approaches. Either the user controls the computations by definition of temporary items and by specifying the way in which the value is computed across the set of entries. The alternative approach means that the computation of certain frequency required functions, such as summation, is built into the system and the user may invoke the function with a single statement or word.

GIS

A set of "automatic" file processing statements is provided which perform certain functions across the entries of a file. Since all entries are included, these statements are usually used on hold files of entries selected in a previous interrogation. The statements may not be used within a LOCATE/EXHAUST block since they provide their own file processing algorithm. Each statement or group of up to five contiguous statements initiates a pass over the input file.

The form of the statements is

```
TOTAL      itemname
AVERAGE    itemname
```

TOTAL builds up the sum of values of item name and AVERAGE computes an average by computing both the sum and a count (see 4.10.2) and then dividing.

Similar facilities exist for computing the sum and average of an item based on the changes of some other items. With all these facilities there is a standard output format for the computer quantity.

A further set of automatic computation is provided by the so called "digit-appended" processing statements which may be used in conjunction with LOCATE/EXHAUST blocks and when the computed quantity is to be included in a user formatted report.

$$\begin{matrix}\text{TOTALn} \\ \text{AVERAGEn}\end{matrix} \quad \left\{\begin{matrix}\text{VARIABLEn} \\ \text{arithmetic expression} \\ \text{itemname} \\ \text{numeric literal}\end{matrix}\right\}$$

where the appended n may be between 0 and 99.

Each time a TOTALn is executed the contents of the quantity specified in the rest of the statement are added to the specially assigned working item. The computed quantity may be output at any time by being referenced in a LIST statement. AVERAGEn is similar except that the average is computed only when referenced in a LIST statement.

Finally in addition to the above special functions, it is also possible to compute file level derived data by use of procedural statements.

MARK IV

Using the Output Content Specification form, five functions, identified as TOTAL, CUMULATIVE, MAXIMUM, MINIMUM and AVERAGE may be specified for computation across the set of entries satisfying the conditional expression specified using the Processing and Record Selection form. Each may be computed automatically for printing at control breaks and/or at the end of the report. The functions may be specified for an item name in the entry or may not be applied to data computed in temporary items.

NIPS/FFS

Using the Output Processor, data items may be summed across the entries in the Qualifying Data File.

With QUIP, the following statement may be used

```
                         ⎧ AVERAGE     itemname             ⎫
COMPUTE tempitem EQ      ⎨ PERCENT     itemname1/itemname2  ⎬
                         ⎩ VARIANCE    itemname             ⎭
```

The quantities computed are printed in a system supplied format with no facility for user over-ride.

It is also possible to compute the sums of the values of one or two items for each occurrence of values of other items. A sum may also be accumulated for remaining values of the items, other than those values specified individually.

The format of the statement is

```
 FOR itemname1  EQ value1   [,value2, ....]   [*]
[AND itemname2  EQ value3   [,value4, ....]   [*]]
 SUM itemname3,  [itemname4, ....]   [HTOTAL]
```

Assuming the AND clause is used, a table is produced with the values of itemname1 on the horizontal axis and values of itemname2 on the vertical axis. Inclusion of the asterisk causes accumulation of the residual sums for each item. Inclusion of HTOTAL causes inclusion of row sums.

<u>TDMS</u>

Both QUERY and COMPOSE permit the use of the functions such as SUM AVERAGE, MAXIMUM and COUNT. In QUERY the functions are applied only to values isolated by a condition expression. In COMPOSE, using the RECAP or CONTENT statements, the statement has a range which implies the range over which the function is computed.

<u>UL/1</u>

To compute values across a set of entries, a special type of procedure called a "file oriented procedure" must be specified. These procedures may invoke "record oriented procedures" (see 4.9.1). Using this facility, computations across a set of entries may be specified for print out at the end of the report. It is also possible to cause intermediate print out at sortkey breaks.

MARK IV

A function called COUNT, analogous to the other five (see 4.10.1) is provided with similar capabilities and restrictions. Further requirements to count the occurrence of values may be specified using temporary items.

NIPS/FFS

It is possible to compute counts using a statement form similar to that used for computing sums, namely

```
     FOR itemname   EQ  value1   [,value2,  ....]  [*]

    [AND itemname2  EQ  value3   [,value4,  ....]  [*]

  COUNT itemname3  [itemname4,  ....]   [HTOTAL]
```

TDMS

The smallest and largest values for any item and the number of occurrences of each in the file may be displayed with the QUERY facility by using the SHOW statement. The user would then normally use the SEARCH facility which allows him to specify a low value and a high value and an integer. A report is then displayed showing values between the limits within a frequency indicated by the integer, for instance every fifth value.

A COUNT statement may be used to obtain a frequency distribution of item values in the set of entries or groups selected.

UL/1

Special reports, called COUNT reports, may be generated by use of a COUNT statement. The frequency of occurrence of any extractable item attribute (or item or entry level derived data) across a selected set of entries may be generated. Two or more COUNT statements may be used to generate reports containing the frequency of co-occurrence of the items or other quantities included in the statement.

An extension of the COUNT report on a single quantity is provided by the INVERT statement. This allows not only a list of values occurring and the frequency of occurrence of each from the selected set of entries, but also the values of some other quantity (such as another item or a record oriented procedure from the corresponding entries. If no other quantity is specified, the value (or values) of the entry identifier is automatically included.

COBOL

Use of the SUM clause in the Report Writer facility automatically causes provision of a temporary item. As each GENERATE Statement is executed, the values of the numeric items named in the sum clause are automatically added to this temporary item. The sum can be printed when required and can also be reset under user control. Using this facility, both row sums and column sums can be generated.

SC-1

Data items may be summed across a file by using temporary data items which are modified in the report description processing for each entry. The temporary items may then be used in a summary section at the end of the report.

4.10.2 Cross entry counting

The facility to compute the frequency of occurrence of values according to specified criteria is frequently provided. If a built-in facility to accumulate sums of values is provided, (see 4.10.1), then facility to accumulate value counts may be specified in a similar way. In some cases the counts may be included in the body of and/or at the end of reports containing computed values. In other cases, count reports have a special format and may not be associated with the reports.

GIS

Using "automatic file processing statements" similar to those for computing sums and averages, (see 4.10.1), it is possible to accumulate a count of existing values for any item. A parallel facility permits incrementing a count each time a value changes from that in the previous entry. The statements used have the formats

```
        COUNT         itemname
     UNICOUNT         itemname
```

The same restrictions on use apply as for TOTAL and AVERAGE. Analogous "digit appended" processing statements, namely COUNTn and UNICOUNTn are also provided for more procedural counting requirements.

COBOL

Counts of the frequency distribution of values across a set of entries in the report must be computed procedurally within the READ/GENERATE loop.

SC-1

Temporary work items may be used to count instances of an item across entries.

4.11 Report formatting facilities

Facilities for formatting reports complement those for specifying what data and derived data may be included irrespective of format. The report formatting facilities are on different parts of a procedural spectrum. At one end the format is completely under control of the system such that the user specifies only what is to be included. Sometimes the system may even compose column headings. At the other end of this spectrum the user has a fairly complete control over format of all content, detail, and peripheral text, such as headlines.

GIS

System formatted reports are provided by use of the LIST statement which may be used to generate a report containing a set of quantities which may be any selection of item values, literals, values from temporary items, or values computed by evaluating arithmetic expressions.

To specify user formatted reports, a REPORT statement must be used together with associated statements which allow the user to control the format and spacing of all parts of the report. A REPORT statement must be used within the LOCATE/EXHAUST block used to pass the file (master file or hold file). Such a facility may be used in either a QUERY subprocedure used for interrogations or in a MODIFY subprocedure used to specify updating (see chapter 5).

MARK IV

The Report Specification portion of the Information Request form is used to specify system formatted reports. With this facility, it is possible to specify sorting and also the computation of the six standard functions (see 4.10.1 and 4.10.2).

User formatted reports require use of at least three forms, namely Processing Record Selection, Output Content Specification, and Title. A Title form is used to specify non system controlled reports (called free form).

NIPS/FFS

QUIP provides a system formatted report facility with some limited user over-ride. It operates in two modes. One, called the paging mode, formats the output into fixed length pages. The other, called the display mode treats the output as one extremely long page and is intended for output on video display scopes.

User formatted reports may be specified using the Output Processor. Default options in this facility allow the user to specify reports which are almost systems formatted.

TDMS

The QUERY facility, for interactive use from a terminal, allows a user to specify system formatted reports containing item values, results of computations, and certain summary derived data. COMPOSE may be used for more complex systems formatted reports but also provides several statements which allow the user complete control over report format.

UL/1

System formatted reports are provided by use of the Automatic Report Generator. This facility allows the specification of sort-keys, heading lines, footing lines and repor titles. The report specification is analysed to decide which of two built-in formats to use. If it is possible to fit all printed quantities from an entry within the page width allowed, a columnar format is used.

To specify user formatted reports, the Complete Report Generator is used. This allows extra statements, not permitted in the Automatic Report Generator, with which the user must place each printed quantity in a two dimensional matrix of which there may be more than one across the width of a page.

COBOL

The Report Writer facility is less procedural than complete tailoring of a report using other Procedure Division facilities. The Report Writer allows the user to define the format of several report blocks which are then used under procedural control in the loop generating the report. Built-in facilities provide for line counting and page counting.

SC-1

User formatted reports are specified using the RDC-1 Report Language statements in the Report Description programs. Default options are included for such features are margin settings and page length.

4.11.1 Item level editing

The editing of an item level quantity, irrespective of its positioning within line, page, or report block may be specified in a number of ways. Sometimes an edit mask is entered for the item at data definition time, sometimes at report specification time. Sometimes it may be specified at both times with the latter overriding the former.

GIS

Before an item value is placed in a report (either user formatted or system formatted), it is converted and formatted in accordance with the decoding and mask specification entered as part of the data definition (see 3.2).

MARK IV

Item level editing may be specified for each item as part of the data definition. It may be over-ridden in a user formatted report by use of an alternative specification using the Output Content form.

NIPS/FFS

Item level editing may be specified for each item as part of the data definition. It may be over-ridden or suppressed in a user formatted report specified using the Output Processor. Using QUIP, the edit masks specified as part of the data definition are always used.

TDMS

In COMPOSE, item level editing may be specified in a MASK statement. The picture forms allowed control placement of decimal point, commas, prefixes, and suffixes. Leading zero suppression may also be specified.

UL/1

Item level editing may be specified as part of a user-formatted report specification only to the extent that the form in which an item value (or a derived value) is printed is determined by the size of the field available to print it. Text may be inserted to the right of or above each printed quantity, and thereby play a role in influencing the format of the printed quantity.

COBOL

Item level editing is controlled by the Picture clauses specified as part of the data definition (see 3.2).

SC-1

Item level editing may be specified as part of the statements which generate the detail lines of the report. Only numeric items may be edited; alphanumeric items are printed as they appear in the data base.

4.11.2 Report body formatting

The formatting of the body of a report includes the placement of item level quantities on a line, the specification of report blocks (both vertically and horizontally) across the paper, and controlling the width and height of a page.

4.11.2.1 Item level placement

In a system formatted report, the user typically has no control over item level placement although he may be allowed some minimum control such as specifying the spaces to be left between adjacent columns in the report.

In a user formatted report, there are two basic approaches to the placement of item level quantities. Either the user must place each quantity in a given location with respect to the page, the line on the page, or some previously defined block within the page. Alternatively he must step through from one item quantity to the next locating each quantity with respect to its predecessor, causing a jump to the next line or block if necessary.

GIS

Using the LIST statement, the user has no control over item level placement. With the REPORT statement, the user must designate lines of the report body as either a detail line or a summary line, using a DETAIL or SUMMARY statement of the form

$$\left\{\begin{array}{l}\text{DETAIL}\\ \text{SUMMARY}\end{array}\right\} \quad [\text{ON conditional-expression}]$$

```
      pp    itemname
      pp pp .... pp itemname
      SPACEn
```

where pp designates the left-most position of the field in which the quantity is to be placed. If there is more than one pp designation, this means that several values for the quantity are expected and are to be printed in the successive locations.

The SPACE statement indicates the number of lines to be skipped before the line designated is printed. The line may be included based on evaluation of the conditional expression.

MARK IV

In the Report Specification portion of the Information Request form, a two digit column is provided in which the user may indicate the number of spaces to be left before each printed quantity. If it is left blank, two spaces are left automatically. The width of columns is determined by taking the longer of the column heading and the edited item. The capability provided in the Output Content Specification form used for user formatted reports is identical.

Using Title Forms, the user has control over the placement of items on the page.

NIPS/FFS

With QUIP, the user has no control over item level placement. Using the Output Processor, a number of statements allow full control over the placement of items, including also the facility to place one quantity or another in a designated location depending on the evaluation of a conditional expression.

The basic statement is of the form

$$\text{LINEn} \qquad \text{pp} \qquad \left\{ \begin{array}{l} \text{itemname} \\ \text{tempitem name} \\ \text{system name} \end{array} \right\}$$

Possible system names refer to the page number and date. To print a variable length item which would overflow a fixed length location on a single line, the following statement is used

```
LINEn     ppl   pp2  itemname
```

where ppl and pp2 designate the right-most and left-most positions on the line.

Facilities associated with that of defining blocks include those for specifying sorting and summary data. If more than one block is allowed across the width of the page, then the user may be able to choose between sorting in which the horizontal direction has priority over the vertical and vice versa. Alternatively one of these two may be built-in, thus not allowing any user option.

GIS

Using the LIST statement, only single line blocks may be defined. However if the data to be printed is too long for one line, it overflows on to the next line. With the REPORT statement, multi-line blocks, one across a page width, may be specified in a LOCATE/EXHAUST block. Blocks which overflow the length of the page can also be specified in this way. To generate a report containing two or more blocks across the width of a page, it is necessary to store entry data in temporary locations and then to print each line partly from the current entry, partly from the temporary locations.

MARK IV

Using the Information Request form, only single line blocks may be defined. However if the data to be printed is too long for one line, it overflows on to the next line. On the Output Content form a column called END LINE may be used to allow two line or multi-line blocks. It is not possible to have more than one block across a page. One block may fill a whole page or more.

NIPS/FFS

In the Output Processor, the user may define multi-line blocks. In QUIP, statistical data maintained by the system is used to determine the format of each line. The LIST statement uses as many lines as necessary to print the items specified. After the first line, each line is indented.

TDMS

In COMPOSE, the CONTENTS ARE statement is the normal way of defining print lines. Multiple uses of this statement are permitted to build up multi-line report blocks. Any one use of the statement may cause printing of more than one line if heirarchical relationships are involved.

UL/1

With the Automatic Report Generator, the data is automatically printed in a single line block if it will fit across the width of the page. Otherwise, a completely different format is used in which each item value is printed on one line (or more if necessary), together with the item description.

TDMS

With QUERY, the user has no control over the item level placement. Using COMPOSE, the CONTENTS ARE and RECAP statements provide the capability to designate the order in which the output quantities appear across the page. Use of the "touch up" facility provided by the SPACE, SHIFT, FEED and PUT statements gives the user more complete control over format.

UL/1

With the Automatic Report Generator, the user has no control over item placement. With the Complete Report Generator he normally declares a two dimensional block into which each item level quantity must be placed. The format of the statement used is

```
     name     ROW     n     COLUMN     n [other specifications]
```

This will cause the quantity named (either a item or computed quantity) to be placed in a field starting on the n th position of the m th line of the block. It may extend to the next quantity on the same row. "Other specifications" include designating the quantity as a sortkey, calling for a new page on a value change, and inserting text before the printed quantity.

COBOL

The user must define print lines in a report group by means of LINE clauses. He may position item values on a print line by means of a COLUMN clause. The Report Writer Control System automatically supplies space fill between all items on a print line.

SC-1

Items may be placed at a given position on the page relative to the left margin, or they may be placed at a given displacement past the previous item. A TAB VERTICALLY statement may also be included to position to a specified line on the page relative to the top of the page.

4.11.2.2 Report blocks

The body of the report normally consists of blocks, where a block, in the simplest case, is a single line of print across the full width of the page. Further possibilities are multi-line blocks, multi-page blocks, and two or more blocks across the width of a single page and blocks wider than the width of a page wrapped over either within the same page or on later pages.

The user of the Complete Report Generator may specify the number of lines in a block using a statement

ARRAY LENGTH integer

If he does not, the length of the block is determined automatically by the highest ROW specification (see 4.11.2.1) allowing that data on that line may overflow on to further lines. He may also specify two or more blocks across the width of the page using the statement

ARRAY WIDTH integer

In this case, the maximum length of a block is eight lines. If an "array width" is not used, the width of a block is taken as the width of the page. If sorting is specified in conjunction with two or more blocks across the width of the page, then the sorting is automatically across the page.

COBOL

The Report Writer feature allows the user to define multi-line blocks (called "report groups"). These blocks are defined in the Report Section of the Data Division and a hierarchical description of the contents of the blocks is used similar to that in the Record Description. Each 01 level "entry" in the Report Section and its subordinate "entries" covers a "report group". A "report group" to be printed must be printed entirely on one page and may not be split across pages. Report groups cannot share a line with other report groups. Report groups placement is directed by LINE and NEXT GROUP clauses (mediated by the PAGE clause).

SC-1

The procedural statements in the report description generate a report block for each entry of the file being interrogated. The size, content, and format of each block are completely controlled by these statements. No restriction as to number of lines or number of pages in a block exists. Blocks within a report may have different formats and any number of different formats is permitted. Since the sorting of the file is performed before these blocks are formatted, the blocks themselves cannot be sorted by the system.

4.11.2.3 Page size control

The width of the page for a report may normally be varied for each interrogation. Control over the length (or height) of the page size may also be possible in order to facilitate generating reports as preprinted forms.

GIS

Both width and length of the report page may be specified in the report specification using the REPORT statement as follows

REPORT WIDTH integer   BODYLINES integer

The BODYLINES clause is used to specify the number of lines in the main body of the report on each page, excluding heading lines and footing lines.

MARK IV

The width and length of the report page may be controlled using the Output Format Specification form. In both cases a number or one character alphabetic code may be entered into a three character field on the form, where the one character code selects one of five standard sizes. A blank field in either case defaults to the installation standard.

NIPS/FFS

In the Output Processor, the user may control both page width and the number of lines between last header and first footing line. This is done with a statement

FORMAT PRINT SPACEn SIZEk, LINESm

where SPACE controls line spacing.

In QUIP, the statement

NLINES = integer

controls whether the page is of infinite length (integer is zero), or a fixed length.

TDMS

In COMPOSE, page width may be controlled by use of SPACE and MASK statements, and page length by use of FEED and PUT statements.

UL/1

The page width may be specified in either the Automatic Report Generator or the Complete Report Generator using a statement of the form

PAGEWIDTH n

If this is omitted a default size of 132 characters is chosen. The page length is built into the system at sixty lines per page. It could be changed only at system assembly time.

COBOL

The page width is controlled only by the way in which the report body is formatted. The page length is controlled by the PAGE clause which defines also the vertical sub-division in which the output groups are presented.

SC-1

The left and right margins used in the report and the page length of the report may be specified by Report Language statements. The margins or page length settings may be modified at control break in the report. A system default for each setting is used if none is specified.

4.11.3 Titles

Titles may be included at the beginning or end of a report and typically consist of lines of text. Capabilities may be provided for including other types of data within a title line.

GIS

In the REPORT statement, title may be written on the first page of the report using a title statement which is of the form

```
TITLE [ON conditional-expression]
pp literal-string [ON conditional-expression]
SPACE n
```

The SPACE statement indicates the number of lines to be skipped on the page before the title is printed. A title may have several lines and may be included or omitted based on evaluation of a conditional expression.

MARK IV

A special form called the Title form must be used to specify a title for a report. Only user formatted reports, namely those specified in the Output Content form, may have a title preceding the report. The title appears on the second numbered page of the report. If the number of lines is less than 12 the title is centered vertically. Otherwise it starts on the top line of the page.

NIPS/FFS

With the Output Processor, title lines may be printed at the beginning of a report following any header lines specified. Title lines are specified by a statement of the form

$$\text{TITLELINEn} \quad \text{pp} \quad \left\{ \begin{matrix} \text{itemnames} \\ \\ \text{text} \end{matrix} \right\}$$

and may be used to include initial values and identifying text at the beginning of a report. Similar facilities are provided by the FINALLINEn statement for the end of a report.

In QUIP, a similar effect can be obtained by conditioning the desired printing with the INITIAL operator.

TDMS

A title may be printed at the top of the first page of a report using a TITLE IS statement. The title is normally centered on the line but using "touch up" facilities, other positioning is possible.

UL/1

A title may be printed at the center of any line on the first report page using the statement TITLE followed by an integer for the line number followed by a literal string which must be shorter than the width of the page. Several title lines may be specified. If the word COLUMN n follows a title specification, then the first character of the string is in that position. Otherwise the string is centered across the line.

COBOL

A title is specified by defining a specific type of "report group" called a Report Heading which is processed only once during the generation of the report, namely during the processing of the first GENERATE statement executed. The Report Heading group is similar to other report groups in the report, and may be placed alone on a page.

SC-1

Titles may be included at the beginning of the report by putting literals in the non-procedural section of the report description.

4.11.4 Heading lines and footing lines

Heading and footing lines are lines of text which may be included at the top and bottom, respectively, of each report page.

GIS

When the LIST statement is used, a set of column headings may be automatically generated from those inputed at the time of Data Definition. However the user may replace these including short literals in the LIST statement preceding each item name. No other heading or footing lines are possible with the LIST statement.

When the REPORT statement is used, heading and footing lines may be produced with statements similar to that used to include a title (see 4.11.3).

MARK IV

Using either the Information Request form or the Output Content form, the column headings automatically generated are those provided at Data Definition time. In both cases the headings may occupy up to eleven different lines on each page, the first and last always being dashed lines to highlight the headings.

The Information Request form allows the user to specify one headline of up to 59 characters which is automatically centered on the top line of each page unless the page is too narrow in which case it is centered on the second line.

If the Output Content form is used, the title form may also be used to include one or more heading lines on each page of the report. It is possible to produce footing lines by printing heading lines at the bottom of the page. Both may be specified using the Title form and free form text.

NIPS/FFS

With QUIP, up to ten heading and up to ten footing lines may be specified by placing a literal character string after HEADERn or TRAILERn. In the page mode, the heading and footing lines will be centered at the top and bottom of each page. In the display mode, the header appears only at the bottom of the infinitely long page.

With the Output Processor, the user can specify HEADERn and TRAILERn lines (where n can range from 0 to 9) and one OVERFLOW line per report. In HEADER and TRAILER lines, the user can specify the printing position of system named variables, such as OPDATE, PAGENO, and CLASSIF. He can also specify the number of lines yet to be printed on a report page and the number of instances of repeating group n to be processed.

In the OVERFLOW line, the printing positions of data items, system named variables, and literals may be specified. The OVERFLOW line prints after the header lines on pages reached by overflow from a previous page. HEADER, TRAILER and OVERFLOW lines may all be conditionally printed on the basis of the value of system named variables.

TDMS

Using COMPOSE, it is possible to include heading lines using a HEADINGS ARE statement. These are effectively column headings and are normally at the top of each page. With the "touch up" facilities, it is possible to place them anywhere on the page.

UL/1

Using either report generator facility, it is possible to include several heading lines and several footing lines which are automatically centered on the line unless a specific starting column is indicated. Values from the most recent entry examined or the result of a computational procedure may be included in heading lines or footing lines.

COBOL

Both heading lines and footing lines are specified by defining specific types of "report groups" called Page Heading and Page Footing. When these are used, a PAGE clause must be included in the Report Description.

SC-1

Up to three heading blocks and one footing blocks may be specified to appear on each page. There is no restriction on the number of lines in either. The heading blocks may include data items as well as literals.

4.11.5 Other embedded literals

Apart from titles, heading lines and footing lines, it is usually possible to specify other literal strings of text for inclusion in the body of a report. Such literals may be item oriented in the sense that they are appended to each instance of an item level quantity appearing. They may also be oriented towards report blocks or to summary data appearing at sortkey control breaks.

GIS

Using the Report Specification facility, in both the DETAIL and SUMMARY statements, the quantity to be printed may be a non-numeric constant. Its inclusion may be conditioned on the evaluation of a conditional expression.

Using the LIST statement, it is also possible to embed literals in output lines.

MARK IV

No other embedded literals are permitted with the Information Request form. However with the Output Content Specification form, the user may include short strings of text when specifying output editing mask to over-ride those supplied at Data Definition time.

NIPS/FFS

Using either QUIP or the Output Processor, literals may be included anywhere in the main body of the report.

TDMS

Using COMPOSE, literal strings may be specified with CONTENTS ARE and RECAP instead of the item names.

UL/1

With the Complete Report Generator the user may append either the item name, item description, or a literal to the printed quantity. The value, and its appendage are formatted in the same report field.

COBOL

Literals may be included anywhere in any type of report group, if required based on evaluation of conditional expressions.

SC-1

Literals may be printed anywhere in a report block or in a summary section.

4.11.6 Control break facilities

Control breaks occur either on value changes in sortkeys or on page changes. In either case various facilities may be provided for special action at such breaks. Summary quantities may be printed (either user formatted or system formatted), special values may be included or special text. It may be possible to jump to a new page on a sortkey break.

GIS

Assuming that the entries which are being formatted into a report have been sorted, a special conditional expression containing the relational operator BR (break) can be used to include summary data at a control break. Other conditional expressions may also be used. However BR is included specially for the control break requirement. The condition is true every time an item (or other quantity) changes value, except for the first time it is tested. There is no explicit or implicit interaction with sorting which must have been specified in a separate sub-procedure.

Skipping to a new page on a control break can be accomplished with an EJECT clause in the associated SUMMARY line specification.

MARK IV

In a system formatted report specified using the Information Request form, the user may identify up to nine control breaks which need not necessarily be identical with sortkeys which are separately specified. The user may identify a control break as a subtitle break or a page break. In the former case the new value of the item is printed as a sub-title after any summary data has been included for the preceding value. If the control break is identified as a page break, then a skip to a new page occurs after printing the summary data for the preceding value. The new value of the item is automatically printed on the new page.

In a user formatted report specified using the Output Content Specification form the identical capability is provided.

NIPS/FFS

In the Output Processor, a conditional expression using the special relational operator CHANGE(S) or CH can be used to perform actions at a change in value of an item in the file or of a temporary item. The actions which may be performed are computing, printing or writing output records. The word COMPLETE causes actions which are similar to those performed with FINALLINE.

In QUIP, the words INITIAL, CHANGES and FINAL provide for conditional computation or printing on value changes. INITIAL and FINAL allow for special action at the beginning or end of a report. CHANGES allows computation and printing between the set of entries with one value of an item and the set with the next.

TDMS

Using COMPOSE, it is possible to include a RECAP statement to specify information to be included at control breaks. The RECAP statement interacts with the SORT statement. For each item included in the SORT statement, there may be several uses of the RECAP statement. A RECAP OVERALL statement may be used to include derive data at the end of a report.

There is no facility provided for including summary data on page changes. The FEED statement facilitates jumping to a new page on a control break.

UL/1

With either the Automatic Report Generator or the Complete Report Generator, it is possible to include derived data at sortkey control breaks. This is specified in a statement of the following type

```
COMPUTE procname  BY SORTKEY n  COLUMN   m
```

This causes the file oriented procedure procname to be invoked each time the nth sortkey changes value.

There is no explicit facility provided for including summary data at page changes although it would be possible to simulate this under certain circumstances by incrementing a user provided line count in a user written procedure and using the line count divided by number of lines per page as a sortkey. Interaction with other sortkeys on value must then be avoided.

It is possible to include the word NEWPAGE with any item (whether it is a sortkey or not although normally it would be). When the item changes value, a jump to a new page occurs.

COBOL

Special "report groups" called CONTROL HEADING and CONTROL FOOTING may be specified for inclusion at control breaks.

SC-1

Data items may be tested to take special action for a given value.

4.12 Interrogating the stored data definition

Although it may be possible to include item schema attributes (see 4.7.1) in user specified reports, special facilities may be provided to generate reports which include only schema type information. Such facilities are usually available to any user, although they may be restricted to privileged use by the data administrator (see Chapter 8).

This capability must not be confused with that of generating a report containing essentially all information in the stored data definition. This facility is normally provided by self-contained systems and is used at the time the file is created (see Chapter 6) and at the time it is updated (see Chapter 8). Discussion here is restricted to facility for selecting and reporting parts of the stored data definition.

GIS

The stored data definition may be printed in a report using a statement of the form

```
DDP [filename]
```

If no file name is included, a list of all file names and synonyms is printed. No facility is provided for selecting data about specific items. The report for a specified file contains file level data, item definitions, item name synonyms, sub-item definitions, group level data, editing specifications and encoding-decoding specifications.

MARK IV, NIPS/FFS

No capability is provided.

TDMS

In both COMPOSE and QUERY, two statements are provided to generate a special report which contains only information from the stored form of the schema (or data definition). The first statement, COMPONENTS, extracts item and group names and the associated numbers of each. The other is DESCRIBE and this extracts the type and validation specifications. Each statement acts on a specified item, group, or set of items and in both cases the report which is generated formats the output in such a way as to clarify the hierarchical relationships.

UL/1

A special division is used to generate a special report which contains item schema information for selected items. This division is called the Bookkeeping Division and the control word identifying it is DESCRIBE.

It is possible to select any item, set of named items or contiguous range of item numbers for inclusion in the report. For each item, the following data is included - item number, name, type, group membership, item description, maximum length, storage format of items and storage format within record.

COBOL

No capability is provided.

SC-1

A system utility job may be used to print data base and data file definitions. No facility is provided for selecting data about specific items but any or all files may be selected by control cards. For every data item, the type, size, security level and indexing mode (if any) are included. The capability applies not only to the central data base definition but also to the data definitions used in and by the application programs.

4.13 Mechanized files

The generation of sub-files from the master file is a capability which falls into two classes, not necessarily mutually exclusive. Either a file generated is for subsequent use by the system in the same run or it is a means of communicating data to other processors such as procedural language programs or special application programs. It is also possible that a system makes no difference between these two classes.

GIS

A procedure may involve up to 16 files, some of which may be HOLD files or temporary files. These may be used as temporary storage during the processing of an interrogation (or update). Alternatively the same kind of files may be saved by the operating system for processing by another GIS procedure or by another processor.

MARK IV

Files may be generated for use outside the system including for a subsequent and separate use of the system itself. No facility is provided for defining temporary use during the processing of an interrogation.

NIPS/FFS

The use of RASP implies generation of a file which may be used as a data file or as input for QUIP or the Output Processor. Several of these RASP produced intermediate files can be used as input for a single Output Processor report. The Output Processor can be used to specify and write files for use outside the system.

UL/1

Files may be generated for use outside the system. There is no capability to generate files for use by the system itself in a later part of the same run.

COBOL

This capability is handled in the Procedure Division and is not a feature of the Report Writer.

TDMS, SC-1

No capability in either class is provided.

4.13.1 Files for system's use

The specification of files for system's own use, here called intermediate files, may be a user option or a user requirement depending on the procedurality of the system. There is a distinction between the user being required to specify the existence of an intermediate file and the user explicitly specifying the way it is used in the process of interrogating a master file. Specifying the existence of a file may be a necessary requirement on the user or it may be an option which allows him to influence the processing of a set of interrogations by assigning more space.

If the user is involved in specifying the way intermediate files are used in the processing of the master file, then again this may be a necessary requirement or it might be an option which allows him to influence what would otherwise be a default situation.

GIS

Depending on the nature of the processing the user may specify the use of intermediate files called HOLD files (see 4.13). If a single master file is being passed and one report generated, then no intermediate file is required. If such a file is required, its data structure is automatically defined without further user specification.

MARK IV

The user must specify temporary files for use by the OS/360 sort only if sorting is indeed required in an interrogation.

NIPS/FFS

RASP produces a sequential file called a "Qualifying Record Table". This file contains supplementary information such as the names of stored Output Processor specifications and the identification of repeating group instances that caused file entries to be retrieved. This file, in conjunction with the retrieved data file, can be used as input to the Output Processor.

TDMS

No capability is provided.

UL/1

The only capability by which the user can influence the processing by specifying extra files is by assigning tape overflow areas for the sort. Sorting normally uses the standard disc working area. If this is inadequate and no more disc space can be assigned then tape overflow areas may be used.

COBOL

See 4.13.

4.13.2 Files for use outside system

The specification of files for use outside the system, here referred to as output files, is a capability which is provided to allow use of the entry and intra-entry selection facilities to generate input to a program which performs a class of processing which cannot be specified with the system's data extraction facilities.

Facilities for specifying output files can be as detailed as those for generating reports. There is a set of capabilities which are essentially independent of whether the data extracted is being formatted in a report or in an output file. Such capabilities as sorting (see 4.6.4) and the inclusion of data from the various levels in the data structure (see 4.7 to 4.9.1) are essentially independent. However the facilities for including summary data or derived data (see 4.10 to 4.10.2) are usually reserved for reports.

Restricting consideration to formatting of the extracted data, it is pointed out in the discussion on generalized extraction features (see 4.6.1) that the concept of system formatted and user formatted holds for output files as it does for reports. However, as in the case of reports, there are also degrees of user and system involvement in the formatting.

In the case of output files, the chief distinction between the two is in the placement of items in the output files logical records. Arrangement of logical records within or across physical blocks is usually handled by the operating system, as is the assignment of the output file to a media type.

GIS

No special provision is made for generating files for use outside the system. All files are formatted according to OS/360 standards, and may be used in any program which uses the same storage structure.

MARK IV

Up to ten output files can be specified for any one conditional expression specified in a Processing and Record Selection form. These files are system formatted and a special report is generated indicating how the system performed the formatting. Each file is formatted according to the operating system convention.

NIPS/FFS

The Output Processor can be used to generate variable length record files for input to other programs. A FORMAT statement is used to supply quantities such as file name, entry size and block sizes. RECORDn statements are used in the same manner as LINEn statements are for reports (see 4.11.2.1). OP conditional expressions (see 4.3) can be used both to select the data items to be included and to position them.

Punched card files are handled in a similar manner within a CARDn statement being used like the LINEn statement for reports. In addition, card files can use TITLECARD and FINALCARD statements which cause punching of user specified first and last cards.

TDMS

No capability is provided.

UL/1

Output files may be either system formatted or user formatted. In either case a file is designated as a COBOL file or a FORTRAN file in view of the different sign conventions for numeric data which are required by the two language processors.

In the case of a systems formatted output file, each physical record contains exactly one logical record and records are also fixed length.

With user formatted output files, the user must declare a logical record length which may be up to 7008 bytes. He may declare a blocking factor to block the fixed length logical records into physical records. He must specify a position in the logical record for each quantity to be included. Entry level derived data is permitted for inclusion in a logical record but not file level derived data.

Whether the report is user formatted or systems formatted, a report is generated indicating the format of the logical records in the output file, the length of the logical records and the blocking factor.

5. UPDATE

Updating a data base is the process of using update data to change values in all entries or selected entries, groups or items stored in a data base. This does not include changing the logical data structure of a data base, the data base validation criteria or security procedures. Within this framework, this chapter discusses the update facilities found in GIS, Mark IV, NIPS/FFS, TDMS, UL/1 and SC-1. Of the systems analyzed, these are the ones that provide generalized update capabilities as contrasted with the host language systems where the user programs his own updating. In the systems covered, changes that alter the logical data structure are made by revising or redefining the data definition of the data base (see 3.9).

The update function requires the use of five kinds of information. These are:

a description of the part of the data in the data base that is to be updated.

the data currently stored in the part of the data base to be updated.

a description of the update data to be applied to the data base.

the update data.

the processing rules to be followed in applying the update data to the data base.

To simplify the discussion in this chapter some terms will be used arbitrarily to refer to these kinds of information. The updating process can apply to data in several files in a data base, and hence the description of that data will be found in file definitions of several files. For convenience, however, this will be referred to as the file definition. Some systems refer to the file being updated as the master file. For brevity it will be referred to as the file or the data file. The data currently stored in the file will be file data. The update data will be called transactions and its description a transaction definition. The processing rules or algorithms that apply a transaction to the data file will be called a transaction program.

An individual transaction is the data that triggers one execution of a transaction program. When data is received via communication lines a transaction may be a "message" in which the message type or part of the message content triggers the execution of one or more transaction programs. When data is entered manually from a terminal, the transaction program to be used may be identified at the beginning of the terminal session or its identification may be included in the transaction. Both of these possibilities also exist in batch processing where the transaction program may be defined or invoked at the beginning of the batch. This invocation will then process a series of update data records each of which is considered a transaction. On the other hand, part of the data in each batched update data record (i.e. transaction) may trigger the execution of one or more different transaction programs.

The first two of these types of information are usually obtained from other system functions. The file definition, which describes the part of the data base to be updated, is the output of Data Definition (see Chapter 3). The data currently in the data base results from File Creation (see Chapter 6) and previous applications of the update function or programming facilities (see Chapter 7). The remaining three types of information are usually directly related to the update function. Systems vary in the format in which each of these may or must be supplied, the time at which they may or must be entered, the acceptable input media for each, and the method used to bring them together. Within a given system there may be more than one technique available to the user for combining these kinds of information. The Sources of update information shown in Figure 5-1 indicate the general approach to combining these kinds of information taken by each system.

In Figure 5-1 transaction definitions are the descriptions of transaction data that allow the system to find it and recognize its attributes. Transaction programs specify the rules by which the transactions (i.e. the update data) are applied to the data file. If these are prestored, it means that in a separate run prior to the update run they were stored in a form and on a medium that makes them accessible to the system.

5.1 User update control

User control of file updating depends on the file and transaction data definitions he can use or specify and the processing rules or algorithms he can specify in transaction programs. The systems studied, often provide more than one mode of updating that sets the framework within which the user specifies the updating to be performed. The update modes provided may differ in the language used, the input media used or the amount of user control allowed. The update modes available are specific to each system.

SOURCE	SYSTEM					
	GIS	MARK IV	NIPS	TDMS	UL/1	SC-1
BATCH PROCESSING MAY USE						
PRESTORED						
TRANSACTION DEFINITIONS	yes	yes	yes	yes	no	yes
TRANSACTION PROGRAMS	yes	yes	yes	yes	no	yes
TRANSACTIONS	yes	no	no	yes	yes	yes
AT THE BEGINNING OF THE TRANSACTION STREAM						
TRANSACTION DEFINITIONS	yes	no	yes	no	yes	no
TRANSACTION PROGRAM	yes	no	yes	no	yes	no
IN THE TRANSACTIONS						
TRANSACTION DEFINITIONS	no	no	no	no	no	no
TRANSACTION PROGRAMS	no	no	no	no	no	no
REMOTE TERMINAL PROCESSING MAY USE						
PRESTORE						
TRANSACTION DEFINITIONS	yes	no	yes	yes	no	no
TRANSACTION PROGRAMS	yes	no	yes	no	no	no
TRANSACTIONS	yes	no	no	no	no	no
AT THE BEGINNING OF THE TRANSACTION STREAM						
TRANSACTION DEFINITIONS	yes	no	no	no	no	no
TRANSACTION PROGRAMS	yes	no	no	yes	no	no
IN THE TRANSACTION						
TRANSACTION DEFINITION	no	no	no	no	no	no
TRANSACTION PROGRAMS	no	no	no	yes	no	no
INPUT MEDIUM FOR TRANSACTIONS						
PUNCHED CARDS	yes	yes	yes	no	yes	yes
MAGNETIC TAPE	yes	yes	yes	yes	yes	yes
MANUALLY OPERATED TERMINALS	yes	no	yes	yes	no	no
COMMUNICATIONS LINES	yes	no	no	no	no	no

Figure 5-1
Sources of update information

GIS

Two modes of updating, UPDATE and MODIFY are distinguished by the data structures to which they can apply changes. In the UPDATE mode, transactions are described and equated with the file data. In this mode update processing can be specified for any of the data structures in the file. In the MODIFY mode, changes can only be made to file items. The items to be changed are selected by the use of interrogation type condition statements.

MARK IV

The user may request file updating on the Transaction Definition Form and/or the Processing and Record Selection Form. The forms are differentiated by the amount of user control allowed. The Transaction Definition Form provides the maximum amount of automatic processing by the system. The Processing and Record Selection Form provides for the maximum user control over update processing. Where the two forms differ, they will be described separately.

NIPS

The basic NIPS updating mode puts the transaction definition, the transaction program, and the transactions into a batch input job stream. There are three languages that can be used to write the transaction program. There is an assembly type language with 81 operators, called POOL. Another update language is the Ordinary Maintenance (OM) language which provides 9 keywords that can be used to specify transaction validation and unconditional changes of file items. The third language, the New File Maintenance Language (NFL) is a high-level, English-like language. It has a repertoire of 23 statements. Unlike OM, these statements can not be mixed with POOL instructions. If a transaction definition and the corresponding transaction program have been stored in the "Logic Statement Library" part of the file to be updated, an update run can be initiated and the transactions entered from a terminal. This type of operation is called Source Data Automation (SODA). Except for special situations where it seems useful to include POOL or OM, this description will be confined to the New File Maintenance Language (NFL).

TDMS

MAINTAIN and UPDATE, the two modes used to update a data base, differ in the functions that can be performed and the input media used. The MAINTAIN, or batch mode, requires the transactions to be submitted in a batch job stream and converted into a TDMS structured file prior to the execution of the update. Transaction programs are entered through on-line terminals. This is the primary update mode and offers features not available in UPDATE. A KEYS ARE clause is used to associate transactions and entries in the file, and a new file results from processing the transactions. The second, or UPDATE, mode allows for a transaction definition, a transaction program and the transactions to be submitted on-line and interactively for a dynamic update of a file.

UL/1

There are two modes called "selective" and "discrete" update. The main difference between them is that in a "discrete" update the user must supply the entry identifier of each file entry to be updated. In a "selective update," the criteria that an entry must satisfy to qualify for updating must be specified by the user. The criteria take the place of the entry identification required by the "discrete" update mode. In either mode the user may define several transaction types at the beginning of a use of the Update Division. These may be invoked by name in the appropriate transactions as they occur later.

SC-1

The update facilities described in this chapter supplement programmer facilities (see Chapter 7) that can be used to specify any data processing. The update facilities use the auxiliary data definitions for files (see Chapter 3). The transaction definitions and transaction programs are processed by a Page Builder program that creates a file in which each entry defines a transaction and its processing.

Transactions may be introduced enveloped by Message Description and Validation Batch Control header and trailer cards.

As part of the transaction definition, the user can specify certain validation criteria that can be checked without reference to the data base. The transactions not rejected by the Message Discrimination and Validation job that applies these criteria are placed in a transaction file. In a separately set up job, they are selected from this file by transaction type or by transaction type and input batch and sorted into file sequence. The user's prestored (i.e. by "Page Builder") transaction program for each transaction type is then used to process each transaction.

An alternate "pipe line" mode is used to completely process one transaction before obtaining another one. This eliminates the use of the intermediate transaction file.

5.2 Data description

5.2.1 Transaction definition

A variety of techniques are used to supply transaction definitions to systems. Sometimes the format and type of transaction that is acceptable to the system is prescribed. In other cases, any type of transaction is acceptable if its transaction definition had been previously stored so the system has access to it. Another technique is to provide a language capability that allows transactions to be submitted to the system in a self-describing form so that the transaction and its definition are intermingled. On the other hand, strict separation of transaction definition and transaction can be found in systems that use the same input medium for both, but require the transaction definition to precede the transaction. The variety of techniques used in transaction definition is not limited to the time and method of its presentation but includes the conventions used to name transaction entries, groups, and items, to identify the corresponding data element in the file, to describe item types (i.e. binary, alphanumeric, etc.) on the update input medium, etc.

GIS

Both update modes may use transactions from "system" or "hold" files. In the UPDATE mode, transactions may also be taken from a "mulrec" file. "System" and "mulrec" files, are defined in a Data Description Table that is created at file definition time (see Chapter 3). A "hold" file is a temporary file that is implicitly defined by the HOLD statement (see 4.1.2) in the interrogation that created it.

MARK IV

Before an update run is initiated, both the file to be updated and the transactions must be described by the user filling in File Definition and Transaction Definition forms and submitting them for processing so they can be entered in the MARK IV "dictionary". The definition of the transaction file may include more than one transaction format. If the Processing and Record Selection facility is to be used in conjunction with the Transaction Definition Form, the transaction file must also be described on the File Definition form.

Each transaction format defined is identified by a Transaction Identifier. No changes in these definitions can be made during an update run. Processing and Record Selection logic, however, can be used to change dynamically the format and/or identifier of a transaction to make it fit an existing definition.

NIPS

The file to be updated is defined in a File Format Table that is created at file definition time from information specified by the user (see Chapter 3). A transaction program is made up of Logic Statements. A Logic Statement may contain both the Transaction Descriptor Cards that specify transaction format and/or the rules by which the transaction is to be applied to the file. There can be a Transaction Descriptor Card for each item in the transaction. It gives the name of the item, its location in the transaction, its data mode (i.e. alphabetic, binary, coordinate, or slphanumeric EBCDIC), and entry and group identification. A separate "Logic Statement" is necessary for each different type of transaction format to be processed during a run.

Ordinary Maintenance Transaction Descriptor Cards provide for checking through the PICTURE, VALUE, RANGE and VERIFY statements. In the PICTURE statement, the user specifies whether the character is alphabetic, numeric, blank, non-blank, special, not special (i.e. alphanumeric) or anything by a code.

The VALUE statement allows the user to specify up to 10 values that are acceptable data values for an alphanumeric or decimal item. Likewise, the RANGE statement allows the user to specify up to 10 ranges within which acceptable data values for an item must fall. The VERIFY statement permits the user to give the name of one subroutine or table that will be called to check the validity of data supplied for a data item.

TDMS

An UPDATE transaction is defined in narrative form as shown in the following typical example.

```
SET EMPLOYEE NUMBER EQ 1000, SALARY EQ 15000 WHERE EMPLOYEE
     NAME EQ SMITH.
```

SET is the system command to affect a change to the value content of file data. In this case, Smith's EMPLOYEE NUMBER and SALARY will be changed to 1000 and 15000 respectively. The WHERE conditional expression selects the file entries to be changed.

When the MAINTAIN facility is used, transactions are placed in a separate TDMS structuref file that was previously defined in the same manner as any other TDMS file.

UL/1

Transaction definitions may be specified at the beginning of a batch of transactions. The transaction definition is considered "temporary" and must be re-entered if needed again in a later run. In both update modes, two methods of transaction definition can be used.

A transaction is defined in narrative form as shown in the following example of a Selective Update:

```
SELECTIVE UPDATE
SALARY EQ 1500, AND EMPLOYEE SMITH EQ
MODIFY RECORD
EMPNO REPLACE WITH 1000
```

where REPLACE is a system command that will change the value of a file item.

A transaction may be defined as a special case of a "macro" as shown in the following example of a Discrete Update

```
MACRO &TRANS (&PARAM1, &PARAM2)

MODIFY RECORD &PARAM1

SALARY REPLACE WITH &PARAM2 * MEND
```

where PARAM1 is the entry identifier, and PARAM2 is the new value for SALARY.

Transactions using this generalized macro would take the following form:

```
&TRANS (76328, 18000)

&TRANS (82571, 16000)
```

SC-1

A fixed format tabular transaction definition with one data element per line is used. Before updating can take place, this transaction definition must be processed and prestored in a "page" by a Page Builder run.

On the Transaction Definition Form, the first three characters of each line must be a unique alphanumeric transaction type identifier. This is followed by a four character Tag which is an identifier code that can be used in place of the name being defined. The next two digits are level numbers like those used in COBOL. They define the data structure of the transaction. This is followed by a one character Type Code that indicates the kind of data element being named on the line. For items, the code identifies the item type as alphanumeric, decimal integer, binary integer, floating point, or a file data name. Codes also identify non-repeating groups (i.e. SC-1 "statements"), repeating groups, and files.

The Size of the transaction data element defined follows the Type Code. The Size of an item is the number of characters it contains. For non-repeating groups, it is the number of items and groups, and for repeating groups or files it is the number of group instances. If the number can not be stated exactly, a V is used, and the user specifies the delimiting characters used in transactions that indicate the end of items, transactions and files in his transaction program.

On the transaction definition form the name of the data element follows its size. The names must start with a letter, contain no more than 31 characters and not use a hyphen as the last character.

Four types of item validation are provided as part of transaction definition. The Type Check validation checks a transaction item to see if it is consistent with one or a specified combination of the system defined character sets (i.e. A through Z, blanks, zero through 9, hyphen, plus or minus, or period). This check may also include a check on any user character sets defined by the ALPHABET statement.

Two restrictions may be placed on the Type Check. The word MUST means that null values (i.e. blanks for alphanumeric items and zeroes for packed decimal, floating point, binary integers or bit strings) are not acceptable, and an integer placed in front of the Type Code limits the checking to the specified number of characters.

The Value Check checks to see that values submitted equal a literal either numeric or alphanumeric, a transaction item value, or an arithmetic expression made up of these. Acceptable multiple values may be specified by joining them with OR's or ranges may be stated by giving the inclusive limits joined by a TO. Numeric values are compared on an arithmetic basis and alphanumeric values are compared character by character using the collating sequence if necessary.

The Total Check specifies the transaction item which when summed should total to the defined item.

The Test Check evaluates a Boolean expression. If a Boolean condition is used, the condition may use the names or identifier codes of transaction items, data item names, literals, arithmetic operators, the relational operators (see 5.3.4), and the logical connectors AND and OR. An expression starts with an item name, followed by a relational operator and is followed by a conditional expression containing literals and/or the names or identifier codes of transaction items that will reduce to a single value at execution time.

An error action can also be set up for each validation check specified. These error actions are reject the transaction, continue checking but do not place it in the accepted transaction file, issue an error message, ignore the error. If error messages are specified, a code may be supplied that causes the corresponding message in the SC-1 Error Surveillance Message Catalogue to be printed. If the user specifies no error action, the entire transaction is checked against its definition, and if any error is found the transaction is not put in the accepted transaction file.

These transaction data definitions must be preceeded by a fixed format message header statement containing a transaction type code, the date, version number, number of sheets, number of days this type of transaction is to be held in a back-up file, transaction name, definer's name, definer's organization code and definer's telephone number. The transaction definitions are also supplemented by two statements specified as part of the transaction program that perform transaction definition functions. One is the ALPHABET statement that allows the user to define his own character set for Type Code checking. It has the following form:

ALPHABET type-code c_1 $[c_2]$ ... ;

The type-code is a one character code that is different from those assigned by the system, and the c_i's are the acceptable characters for this Type Code.

5.2.2 Files used

Descriptions for both the file that is to be changed and the transactions with which it is to be changed must exist or be implied before updating can take place. In the systems analyzed, the description of the file to be updated was stored at data definition time (see Chapter 3) and is available to the system. In the systems studies, even the systems that provide for updating null files, assume that the system's file definition process has already been performed for the file that is to be updated.

In data base management systems the need exists for many kinds of multiple file operations. Systems may allow the transactions to be entered into two or more files in a data base, or they may allow values computed from transactions to be entered into more than one file. In addition to using a transaction file to change data in two or more data base files, it is sometimes necessary to use data from two or more transaction files to update a single data base file. When this condition is complicated by the transactions being on files foreign to the system, it may be necessary to provide an assembly language preprocessor to convert the foreign files to an acceptable system transaction file.

GIS

A maximum of 16 files may be used in a single MODIFY "subprocedure". The transaction files may be "system" or "hold" files, but the data files to be updated must be "system" files. In an UPDATE subprocedure, a single "system" file is updated from a single transaction file. The transaction file may be a "system", "hold" or "mulrec" file.

MARK IV

When the Transaction Definition Form is used, only one transaction file can be used to update one data base file.

When Processing and Record Selection is used, one data base file can be updated by using one transaction file and information from one to nine other data base (i.e. "coordinated:) files.

When the two modes are combined, the user can specify processing that will update any or all of the "coordinated" files.

The opening of files is performed by the OS/360 operating system. File parameters such as blocking factors entered on MARK IV forms override any conflicting parameters in OS/360 Job Control Statements.

NIPS

Only one transaction file can be applied to one data base file at a time. Information from the transaction file and the data base file can be written out on up to five "auxiliary" files. The data file is opened by placing its name in the File Maintenance Control Card.

TDMS

In UPDATE, transactions update an in place data base file. In MAINTAIN, the transaction file is converted to a TDMS structured data base file prior to execution of the update. In this mode, a new data base file is produced as output. The opening of files is performed by the ADEPT operating system. When the opening has been completed, TDMS supplies the TDMS name of the file to the user.

UL/1

Only one transaction file can be applied to one data file at a time. Files are opened by being named in the UPDATE statement.

SC-1

The user must open each data base file to be used in the update run with a TAP statement as follows:

$$\text{TAP-identifier} \quad \left[\text{file-name.} \begin{bmatrix}\text{INPUT}\\ \text{UPDATE}\end{bmatrix}\right] \quad \ldots;$$

The INPUT option indicates the file is opened for reading only. UPDATE allows changes to be made in the file.

Transactions must be sorted in the same order as the file to be updated. The user indicates the correspondence between transaction and file sort keys by:

$$\text{SPECIFY-TARGETS} \quad \left[\text{(file-name)} \; \left[\text{(trans-item data-item)}\right] \ldots)\right] \ldots$$

Core buffers may be established for groups from as many data base files as core capacity allows.

All transactions are prestored in a transaction file. Transactions are entered into this file either individually or in batches.

In a single update run the user may process as many types of transactions as may be sorted on the same items. For such transaction types, the user may ask for the processing of all transactions of a type or for specifically identified batches.

5.3 Transaction program definition

Both the format and the content of transaction programs varies widely among the systems analyzed.

5.3.1 Transaction program format

The formats used include fixed tabular forms, narrative form, and computer procedural language forms. Sometimes more than one form will be used in a system.

GIS

In both UPDATE and MODIFY, transaction programs are written in narrative form, punched into cards, and read into the system. The transaction definition must have been entered into the system previously.

MARK IV

When either the Transaction Definition Form or the Processing and Record Selection Form is used, the transaction definition and transaction program specification are combined on a single tabular form.

NIPS/FFS

The Logic Statement is prepared on punched cards. If transactions are used in the update, the Transaction Description Cards are placed at the beginning of the Logic Statement. The New File Maintenance Language statements that specify the update processing follow the Transaction Descriptor Cards.

TDMS

In the UPDATE mode, the transaction program is combined with the transaction definition and the update data in a narrative form. (see 5.2.1)

In MAINTAIN the transaction program is entered from a terminal as in the following example:

```
KEYS ARE EMPLOYEE NAME

SET EMPLOYEE NUMBER = NEW EMPLOYEE NUMBER

SET SALARY = NEW SALARY
```

In this case all employees having entries in the transaction file will get new employee numbers and salaries if such data is supplied by a transaction.

UL/1

A narrative form of transaction definition is used. It includes both the transaction program and the update date (see 5.2.1).

When the "marco" transaction definition is used, the transaction program is usually included in the "macro" definition and the data is provided as parameters when the "marco" is invoked.

SC-1

The Message Processing Language statements in which the transaction program is specified are punched on cards. They are preceded by a fixed format Processing Header statement.

5.3.2 Data mapping

Some method is needed to equate transaction group and item names to the file group and item names to which they correspond or are related. One method is to use file group and item names in the transaction description for corresponding data elements. This may not be possible when data files, created for other purposes, are used as transactions or when the relation between transaction and file data elements is indirect.

GIS

In UPDATE the correspondence between file and transaction groups is established in the transaction program by a STRUCTURE statement which has the following form:

```
STRUCTURE file-group-name FROM transaction-group-name
```

The correspondence between transaction and file items is established by the EQUATE statement which takes the following form:

```
EQUATE

file-name TO transaction-name

[file-name TO transaction-name]

.
.
.
END EQUATE
```

If a transaction item name is not included in this list, it is automatically equated to a like named item in the file unless the like named file item has been excluded by an EXCEPT statement of the following form:

```
EXCEPT file-item-name [,file-item-name] ...
```

In MODIFY the mapping is programmed by the user. Item correspondence is obtained by the CHANGE statement (see 5.3.5.3) and computational statements.

MARK IV

Item mapping is obtained by using file item and group names on the Transaction Definition Form or the Processing and Record Selection Form.

NIPS/FFS

For Ordinary Maintenance (OM), transaction items are mapped into file items by using file item names on the Transaction Descriptor cards.

When using the New File Maintenance Language (NFL), transaction items and groups are mapped into the data file by MOVE, ATTACH, or COMPUTE statements (see 5.3.5.3).

TDMS

Transaction items and groups are mapped into the data file names by using the file names in the transaction definition.

UL/1

Transaction item and group names must be mapped into the data file names by using the file item names and, for multiple valued items, the ordinal position of the desired value in the transaction definition.

SC-1

Except for file sequencers (see 5.2.2) the user equates transaction items with file items by SET statements (see 5.3.5.1) in his transaction program that move the transaction item value, modified by an arithmetic expression if desired, to the file buffer area.

5.3.3 User control over data access

The amount of user control over data access varies. The accessing of transactions and the file entries with matching identifiers may be done automatically with no user specification. Systems that provide this type of automatic transaction and entry access usually also automatically write out the updated entries. On the other hand, the user may be given complete control over data access by being required to specify each step of the data access process. This may include instructions for individually reading each group contained in a transaction from the input medium. Similarly, the user may be required to locate, read and after the update processing has been performed, write out the entries by using instructions that operate on data elements below the entry level.

GIS

In UPDATE, transactions are retrieved by the system as needed. File entries and groups are retrieved on the basis of matching identifiers and returned by the system when update processing is complete.

In MODIFY, the user programs transaction and file entry and group retrieval and selection using the same statements used for interrogation (see 4.3). For example, the following statements will add one to the age of all males in the personnel file called PERSFILE.

```
MODIFY PERSFILE
LOCATE PERSON
WHEN SEX = 'M'
INCREASE AGE BY 1
EXHAUST PERSON
END PROCEDURE
```

MARK IV

Transactions are retrieved automatically. When the Transaction Definition Form is used, transactions are automatically matched to file entries on the basis of entry and group identifiers. When the Processing and Record Selection Form is used, each file entry is automatically retrieved and checked for conforming to update conditions. In both cases, if an indexed sequential file is being updated, only the changed entries are automatically written out to the file. If a sequential file is being updated, all file entries, whether or not they were changed, are written out to a new file.

NIPS/FFS

When OM is used, transactions are retrieved automatically. File entries and groups with matching identifiers are retrieved and returned automatically.

When NFL is used, the principal items of an entry (i.e. "fixed set") are retrieved if the file contains an entry with an identifier that matches the transaction identifier. Groups are retrieved through the execution of one of the following types of statements:

```
LOCATE SET operand, EXIT label

STEP SET operand, EXIT label
```

The LOCATE statement retrieves the first instance of the repeating group containing the operand. STEP retrieves the next instance of the repeating group containing the operand. In all group retrieval statements, the exit is taken if no instances of the group exist or if all of them have been retrieved.

```
          ( [TO]    FIRST )
POSITION  { [TO]    NEXT  }  operand,  EXIT label
          ( [AFTER] LAST  )

or   POSITION   [TO] operand1  [IN] operand2,   EXIT label
```

For the FIRST and NEXT options, the indicated instance of the repeating group containing the operand is retrieved. For the LAST option the EXIT is ignored, and the next action for the group will take place after the last instance of the group. An appropriate action might be the insertion of an additional instance of the group or the deletion of all instances of the group.

The second POSITION statement selects an instance of a repeating group in which operand1 and operand2 are equal. The "fixed set" and all instances of groups are automatically returned to the file at the completion of their processing.

TDMS

In UPDATE, transactions are read automatically from a manually operated terminal. File entries are retrieved automatically on the basis of matching identifiers. Before releasing an updated entry to the system, a user may request a variety of information, such as the actual number of changed values, to assure himself that the updated entry should be put in the file.

MAINTAIN writes a new file using transactions, that have been put into a TDMS structured file, and another TDMS data file as input. All file reading and writing is done automatically.

UL/1

All transactions, data file entries and groups are retrieved automatically. Changed entries and groups are returned to the file automatically.

SC-1

The GET statement is used to retrieve each repeating group instance at each data structure level required for the transaction processing. Separate GET statements are used to retrieve group instances from the transaction and from the data file. The syntax of the GET statement is:

```
GET {entry-name} IN file-name [WITH conditional-expression]
    {group-name}
[EOF [error-message-code] [error-action-code]
[SKIPTO label]];
```

The entry-name option retrieves the principal items of the entry. A GET statement without a WITH conditional expression puts the next sequential group from the file into a buffer area in which other SC-1 statements may operate on it. When the WITH conditional expression is included, the next group that meets its condition is put in the buffer.

To facilitate the retrieval and processing of groups from transactions the following loop control statements are provided:

```
label DO transaction-group-name;
         .
         .
         .
      ENDDO label;
```

5.3.4 Update data selection

Data selection is the identification by the user of the part of the file that is to be changed by a transaction. The most general way in which this can be done is to state the logical relations that must be satisfied before a transaction is applied to the file. In recognition of this, the same language used to specify conditional expressions in the interrogation function (see 4.3) may be used to specify the criteria.

A second method is for the system to match the transaction to the file on entry and, if necessary, group identifiers. When the identifiers match, the specified processing is performed with no further need for the user to specify data selection criteria.

The third method acts on all file data. An example of this type of updating can be as simple as adding one to the current year for every entry in the file. On the other hand, it may apply the same complex transaction to all file entries. End of year accounting changes can be of this type. Sometimes these changes are applied not to all the entries in the file, but to those entries selected by one of the previous methods.

GIS

In the UPDATE mode, transactions are matched with the file on the basis of entry or group identifiers. Further selection conditions based on data content and relationships in both the transaction and the file can be imposed by using the IF, ELSE and IN ANY CASE statements. These statements take the form:

```
IF conditional-expression

statement-group-1

ELSE

statement-group-2

IN ANY CASE
.
.
.
```

In the MODIFY mode, the user specifies the conditions for data selection by using the same statements used for interrogation (see 4.3).

MARK IV

When the Transaction Definition Form is used, the system selects the file entries and groups that match the transactions on the items that have an M action code. If the Processing and Record Selection Form is also used, its conditions are applied to the entries or groups selected by the Transaction Definition Form. When the Processing and Record Selection Form is used alone, selection is performed in the same way it is for interrogation (see 4.3).

NIPS/FFS

When using OM, the system selects file entries and groups that match transactions on the transaction items the user has identified as corresponding to the file identifiers.

When NFL is used, the IF statement can be used to select file entries and groups to be updated. The IF statement has the following general syntax:

IF operand1 operator operand2 [,operand3] ...

The operators are:

```
      ⎧ EQ      ⎫⎫
      ⎪ NE      ⎪⎬ These 3 operators may be followed by
      ⎪ BETWEEN ⎪⎭    multiple operands
NOT   ⎨ LT      ⎬
      ⎪ GT      ⎪
      ⎪ LE      ⎪
      ⎪ GE      ⎪
      ⎩ TABLE   ⎭
      { PICTURE } [NOT]
      { BIT     }
```

The following IF test statements may also be used:

```
IF operand [BIT] [NOT] {ON }
                       {OFF}

IF [NOT] NEW RECORD

IF OVERFLOW [NOT] {ON }
                  {OFF}
```

TDMS

The conditional expression that is used in the interrogation function (see 4.3) is used in both UPDATE and MAINTAIN to select data from the file. An additional clause is available in the MAINTAIN mode. KEYS ARE allows the user to give up to three items in each file that are used as identifiers for matching transactions and file entries. In this mode the WHERE conditional expression is secondary to the identifier and is used to refine the selection.

UL/1

In "selective" updating, the user must give the "update criterion" using the same form of conditional expression as is used in interrogation (see 4.3).

For "discrete" updating, the user must give the value or values of the identifier of an entry to be modified or deleted.

SC-1

The WITH conditional expression of the GET statement (see 5.3.3) is used to select file groups to be updated. The syntax of the WITH clause is:

$$\left[\text{WITH relation1}\left[\left\{\begin{matrix}\text{AND}\\ \text{OR}\end{matrix}\right\}\quad\text{relation2}\right]\ldots\right]$$

where relationi is:

$$\text{operand1 IN file-name operator}\quad\left\{\begin{matrix}\text{literal}\\ \text{operand2}\\ \text{operand3}\end{matrix}\right\}$$

Acceptable operators are EQ, NE, GE, GT, LE, and LT. Operands 1 and 3 are the names of items in the file. Operand2 is the name of a transaction item.

5.3.5 Data base changes

What the user can change in a data base and how he specifies it, varies. Figures 5-2 through 5-4 show the names of system facilities that perform several kinds of data base changes in the systems studies. A "none" in these tables indicates that the system provides no special facility to perform the function. In Figures 5-2 through 5-4, each system is looked at from the user's point of view in terms of what he can do using the facilities provided by the system in the way they were intended to be used. In Figures 5-2 and 5-3 entries and groups are considered inserted only when, from the user's point of view, data at these levels has been put into the file using system facilities designed for that purpose and can now be retrieved and changed. For groups and entries it means that group and entry instances with previously unused identifier values are now available to the user. Likewise the deletion of instances of groups or entries means that, using the facilities of the system the file has been changed so that the user no longer has access to groups or entries with identifier values that he previously could use. Some systems allow the user to change the values of items used as group and entry identifiers. In sequential files, this usually requires changing the position of entries on the storage medium. Systems vary with respect to the need to change the physical location of a group under such circumstances. For other types of files, it may require changing both the position on the storage medium and the indexes for the file so it will reflect the new identification of the group or entry.

Entry and group replacement are not covered in Figures 5-2 and 5-3. They are dealt with as entry and group level changes (see 5.3.5.1 and

SYSTEM	SYSTEM FACILITY	DATA CHANGE OPERATORS FOR ENTRIES		
		INSERT	DELETE	IDENTIFIER CHANGES
GIS	UPDATE	INSERT STORE INCLUDE INCLUDEF	DELETE REMOVE	none
	MODIFY	none	none	none
MARK IV	Transaction Definition Form	C "record" action code	D "record" action code	R, P "record" action code
	Processing and Record Selection Form	none	System DELETE flag is set	R[1]
NIPS	New File Maintenance (An English-like language)	2	DELETE RECORD	MOVE
	POOL (An assembly type language)	2	DDR	MCT MCW
	Ordinary Maintenance Language (Control card key-words)	GENERATE	none	none
TDMS	UPDATE and MAINTAIN	ADD	REMOVE	none
UL/1	DISCRETE	INSERT	DELETE	none
	SELECTIVE	none	DELETE	none
SC–1		SET's followed by ADD or INSERT	GET followed by DELETE	none

[1] This is equivalent to the COBOL MOVE.

[2] Entry insertion is automatic if the transaction identifier does not match on entry identifier in the file.

Figure 5-2
Entry level data change operators

SYSTEM	SYSTEM FACILITY	DATA CHANGE OPERATORS FOR GROUPS		
		INSERT	DELETE	IDENTIFIER CHANGES
GIS	UPDATE	INSERT STORE INCLUDE INCLUDEF APPEND	DELETE REMOVE	none
	MODIFY	none	none	none
MARK IV	Transaction Definition Form	I "segment" action code	E "segment" action code	none
	Processing and Record Selection Form	none	System DELETE flag is set	R[1]
NIPS	New File Maintenance Language (An English-like language)	BUILD SUBSET[2]	DELETE SET DELETE SUBSET	MOVE
	POOL (An assembly type language)	BSS[2]	DSB DSC	MCS
	Ordinary Maintenance Language (Control card key-words)	GENERATE	none	none
TDMS	UPDATE and MAINTAIN	ADD	REMOVE	none
UL/1	DISCRETE	none[3]	DELETE	none
	SELECTIVE	none	DELETE	none
SC–1		SET's followed by ADD or INSERT	GET followed by DELETE	none

[1] This is equivalent to COBOL MOVE.
[2] This sets up an empty group. Data is inserted by item level changes.
[3] Accomplished by inserting component items.

Figure 5-3
Group level data change operators

SYSTEM	SYSTEM FACILITY	DATA CHANGE OPERATORS FOR ITEMS		
		INSERT	DELETE	VALUE CHANGES
GIS	UPDATE	INCLUDEF REPLACE REPLACEF CHANGE	ERASE	INCLUDE REPLACE REPLACEF CHANGE INCREASE DECREASE MULTIPLY DIVIDE
	MODIFY	CHANGE	ERASE	CHANGE INCREASE DECREASE MULTIPLY DIVIDE
MARK IV	Transaction Definition Form	none	none	P, R, B, A, or S "field" action code
	Processing and Record Selection Form	none	none	R, +, –, *, /
NIPS	New File Maintenance Language (An English-like language)	ATTACH	DELETE FIELD[1]	MOVE COMPUTE
	POOL (An assembly type language)	MVF	CLR[1] CVF[1]	MAL, MNU, MAC, MNC, ADD, SUB, MPY, DVD, GENERATE
	Ordinary Maintenance Language (Control card keywords)	none	none	GENERATE
TDMS	UPDATE and MAINTAIN	SET	FAILS	SET
UL/1	DISCRETE	INSERT	DELETE	REPLACE
	SELECTIVE	INSERT	DELETE	REPLACE
SC–1		none	GET followed by SET TO NULL and REPLACE	GET followed by SET and REPLACE

[1] Variable length items are physically removed. Fixed length items are set to null values.

Figure 5-4
Item level data change operators

5.3.5.2) where the discussion points out that in many cases they are really successive applications of deletion and insertion. In Figure 5-4, although items are always a part of an entry, or a group or of both, changes in item values, except for changes in the value of group and entry identifiers, which are treated separately, are considered item level changes. Some systems provide a special representation for null value items. In Figure 5-4, an item is considered inserted when it is changed from the system's null value to a data value. An item is considered deleted when it is changed from a data value to the system's null value. If the user must set up his own data coding scheme to provide for null values for items, Figure 5-4 does not show this as a system capability for item insertion and deletion.

5.3.5.1 Entry level changes

Entry level changes imply transactions that take into account all the data in a file entry. An entry level change can be a transaction that identifies an entry to be removed from the file. When such a transaction is processed, it will no longer be possible for a user to retrieve or update any items or groups associated with this entry. When a transaction inserts a new entry, the user has access to an entry identifier not previously available to him from which he can now retrieve data and against which he can now process additional updates. In some systems entry level changes require transactions that contain complete entry level data for an entry identification that is already in the file. This may be processed by deleting the entry in the file and adding the entry from the transaction. It is also possible to use transaction entry data to change the file entry only where it differs from that found in the transaction, to change only null-valued file items from the corresponding transaction values, or to replace file values with all nonnull transaction values.

GIS

There are three statements that can be used to insert entries into a file.

$$\left\{\begin{array}{l}\text{INSERT}\\\text{STORE}\\\text{INCLUDE}\end{array}\right\}\ \text{data-file-entry-name;}$$

When there is no matching entry identifier, all three statements insert the transaction as an entry. The insertion includes all the subordinate groups contained in the transaction. When there is a matching entry identifier, STORE and INCLUDE will insert subordinate groups for which, there are no matching groups in the matching file entry. For INSERT, the existence of a matching entry identifier produces an error condition and no updating takes place. No updating also takes place, but with no error indication, when the STORE statement is used with a transaction in which the entry and all group identifiers match.

Entry level changes can be made by two statements.

$$\begin{Bmatrix}\text{INCLUDE}\\ \text{REPLACE}\end{Bmatrix}\ \text{data-file-entry-name;}$$

INCLUDE will replace file entries and subordinate groups with the transaction containing matching entry and group identifiers. Unmatched subordinate groups will remain in the file. The entire file entry and all subordinate groups are removed when a REPLACE statement is used with a transaction that has a matching entry identifier. The transaction with whatever subordinate groups it may include is entered into the file.

Entries may be deleted by using

$$\begin{Bmatrix}\text{DELETE}\\ \text{REMOVE}\end{Bmatrix}\ \text{data-file-entry-name;}$$

Either statement deletes the entry and all its subordinate groups. REMOVE and DELETE differ only in that DELETE produces an error message in case the entry is not in the file.

MARK IV

New entries can be inserted in the file being updated if a "create transaction" action code is put on the Transaction Definition Form. If this has not been done, transactions that do not match an entry in the file will normally be rejected. If requested by the user, rejected transactions will be written on a "transaction reject file" that can be printed or otherwise processed after the completion of the update run. This normal rejection of unmatched transactions can be overridden if the user indicates he wants the DEFAULT CREATE option on the Transaction Definition Form. When this is done, an unmatched transaction of this type, creates a new file entry with the minimum storage space required by the File Definition Form. If the "default create" occurs as the result of a transaction that updates a lower level group, the parent group is also created with null values for its data items.

On the Transaction Definition Form, a file entry may be deleted if the delete operator, D, is used as the "action code" for the type of transaction that is to initiate the deletion. In Processing and Record Selection, a delete flag can be set for each file, the deletion takes place as the new copy of the file is written. In indexed sequential files, a logical record delete byte is set.

NIPS

When a transaction identifier that does not match a file entry is encountered, the transaction is automatically added to the file. At this time a "new record" flag is set. Its existence can be tested by the NFL conditional expression IF NEW RECORD. Entries can be deleted with a DELETE RECORD statement.

TDMS

ADD ENTRY and REMOVE ENTRY statements are available in both the MAINTAIN and UPDATE facilities. In both facilities when an entry is to be removed, the REMOVE ENTRY statement will contain a WHERE conditional expression that supplies the identifier of the entry to be removed. In UPDATE, the ADD ENTRY statement is followed by the data for the new entry.

MAINTAIN writes a new file using transactions that have been put into a TDMS structured file and a TDMS data file as input. For those entries in the transaction file for which there is no entry in the data file with matching identifiers, the ADD ENTRY statements in the transaction program specify how the transaction is to be put into the new file written by MAINTAIN.

UL/1

The statement for inserting an entry using "discrete" updating is shown in the following example:

```
UPDATE filename1, filename2 DISCRETE UPDATE INSERT RECORD
complete-entry-data *
```

The statement for deleting an entry using "discrete" updating is shown in the following example:

```
DELETE RECORD 23641
```

The statements for deleting an entry using "selective" updating are shown in the following example:

```
UPDATE CRITERION SEX EQ FEMALE *

DELETE RECORD *
```

SC-1

When adding a new group at the end of a file or inserting a new group, the new group must be built with a series of SET statements in a buffer area. The buffer area can be thought of as similar to a COBOL record area.

The SET statement syntax is

$$\text{SET file-item-name IN file-name} \begin{Bmatrix}\text{expression}\\ \text{NULL}\end{Bmatrix};$$

The expression is an arithmetic expression that may use both file and transaction item names as variables. The expression may, in effect, equate transaction item names with file item names. After it is built, such a group is added at the end of a file with

ADD entry-name IN file

To correctly insert a group in a file, the file must first be properly positioned by using a GET statement (see 5.3.3) to put the group that is to follow the new group in the file's buffer area. The new group is then inserted with

INSERT entry-name IN file

To delete a group, the group is first retrieved and put in a buffer area with a GET statement (see 5.3.3). It is then deleted with

DELETE entry-name IN file

5.3.5.2 Group level changes

Group level changes imply a transaction that takes cognizance of all the items or subordinate groups within a group. As with entries, a group level transaction may supply only the identification of a group that is to be deleted. Once such a transaction is processed, the user can no longer retrieve data from nor update data in this group or its subordinate groups. A group level change may insert a group in an entry. When it is processed, it makes available to the user group, which was previously not available to him, from which he may now retrieve data or in which he may update data. As with entry level changes, some systems will process group level transactions that contain identifiers that match the identifier of a group already in the file. It is also possible to treat such a transaction as a simple deletion of the file group and insertion of the transaction group. In other cases the processing of replacement groups may be as complex as that described for entries.

GIS

All the statements that operate at the entry level also operate at the group level. The handling of higher level and subordinate groups in INSERT, STORE, INCLUDE, and REPLACE has different implications for groups than for entries. For instance, INSERT and STORE will insert missing higher level group identifiers.

Since STORE leaves file groups unchanged when it is used with a transaction that has matching identifiers, it can be used at a high group level to insert groups at lower levels while leaving existing file groups unchanged. INCLUDE, on the other hand, replaces matching groups, inserts subordinate unmatched transaction groups and leaves unmatched subordinate file groups in the file. REPLACE deletes all file groups subordinate to the matching group and inserts the transaction group with any accompanying subordinate groups. If this is applied at two different hierarchical levels, an entry with an incomplete hierarchy can be created.

MARK IV

Groups may be inserted in fixed length entries if, at file definition time, an empty group of the type to be inserted was created and if during a previous dictionary maintenance run an insert group transaction was defined by putting an "insert" or "default insert action code" on a group Transaction Definition. If a group is defined for insertion of variable length data file entries, MARK IV will enlarge the entry to accomodate the new group.

Any group whose identifier is matched by a "deleting" transaction may be deleted when using the Transaction Definition Form. Groups may also be deleted when using Processing and Record Selection by setting the system "delete flag". Groups may not, however, be inserted when Processing and Record Selection is used alone, but may be inserted when the two forms are combined. Identifiers may be changed with either the "replace" or arithmetic operators when using Processing and Record Selection.

NIPS

The statement

BUILD SUBSET operand

creates an instance of a repeating group (i.e. "SUBSET") preceding the currently active group instance. Subsequent group instances are pushed down. If the repeating group is not active, the new group is added at the end. This statement creates the space and activates the new group instance. No data is moved into the new group instance. This is done with item level statements. Repeating groups are deleted by

DELETE SET operand

Instances of repeating groups are deleted by

DELETE SUBSET operand

TDMS

Repeating groups are added or removed a group instance at a time by using the ADD and REMOVE commands in conjunction with conditional expressions that will isolate the desired group instance. These statements are similar to the ADD ENTRY and REMOVE ENTRY statements except that the group name is used in place of the word ENTRY. The only other difference is that in MAINTAIN when the ADD is used for a repeating group there must be a matching entry in the file to which this group can be added.

UL/1

The syntax for deleting a group using "discrete" updating is shown in the following example:

```
MODIFY RECORD 23610 EDUC DELETE
```

If EDUC is a group name, all groups subsumed under this group in RECORD 23610 will also be deleted.

To delete the EDUC group and all subsumed groups from all records in the file for which SALARY is less than or equal to 15,000, "selective" updating is used as follows:

```
UPDATE EMPFILE
SELECTIVE UPDATE
UPDATE CRITERION
SALARY LE 15000 *
MODIFY RECORD EDUC DELETE *
```

SC-1

Since a entry is a compound repeating group, group level changes are handled in the same way as entry level changes (see 5.3.5.1).

5.3.5.3 Item level changes

Item level changes occur when less than complete group data is contained in a transaction. The items to be changed may be entry or group level items. The amount of identification required to locate the item depends on its placement in the entry. Entry identification is sufficient for entry level items. Both entry and group identification are needed to locate a group level item.

For character string items, changes to individual characters or to sub strings of characters within an item are a special type of value change. Another special type of value change occurs when an item value changes to or from an established system null value. In some systems this is referred to as the deletion or insertion of an item. The physical insertion or deletion of items usually takes place only when groups or entries are stored in variable length physical records.

Some facilities will either inform the user of the effect of changes being made or allow him to supplement the conditions contained in his data selection (see 5.3.4) with another method of stating conditions on item values that must be met before changes are made. An example of supplemental information is the ability of a system to notify the user when the value put in an item is the same as its previous value.

GIS

In the UPDATE mode all item level changes are confined to value changes, including null (i.e. "absent") values, for transaction entries or groups with identifier items that match file entry or group identifiers. These are performed by

$$\left\{\begin{matrix}\text{INCLUDEF}\\ \text{REPLACEF}\end{matrix}\right\}\ \text{data-file-name}\ \left\{\begin{matrix}\text{entry-name}\\ \text{group-name}\end{matrix}\right\}\ ;$$

which use nonnull (i.e. not "absent") transaction entry or group items to change the value of corresponding file items. In case there is no group or entry identifier match, REPLACEF will indicate an error condition and no updating takes place. For similar unmatched conditions, INCLUDEF will insert the transaction into the file with higher level group identifiers supplied. The definitions used for Figures 5-2 through 5-4 treat this as an entry or group level insertion.

In UPDATE and MODIFY, changes to data file items can be made with any of the following statements

$$\text{CHANGE}\ \left\{\begin{matrix}\text{file-item-name}\\ \text{temporary-variable-name}\end{matrix}\right\}\ \text{TO}\left\{\begin{matrix}\text{arithmetic-expression}\\ \text{string-expression}\end{matrix}\right\}$$

$$\left\{\begin{matrix}\text{INCREASE}\\ \text{DECREASE}\\ \text{MULTIPLY}\\ \text{DIVIDE}\end{matrix}\right\}\ \left\{\begin{matrix}\text{file-item-name}\\ \text{temporary-variable-name}\end{matrix}\right\}\ \text{BY arithmetic expression}$$

$$\text{ERASE}\ \left\{\begin{matrix}\text{file-item-name}\\ \text{temporary-item-name}\end{matrix}\right\}$$

Since CHANGE can be used to change an item value from the system's null value "absent" (i.e. not blank or zero) to a data value, it may be thought of as a way to insert an item. Likewise, ERASE which converts an item to the null value can be thought of as a way to delete items.

MARK IV

The Transaction Definition Form provides codes for indicating that item values in file entries or groups are to be replaced by the transaction value, replaced only if the transaction value is non-blank, or replaced with blank or zero If the items are numeric, codes can be used to add or subtract transaction values from item values. In Processing and Record Selection, item values may be changed by the usual arithmetic operators and a replace operator that is the equivalent of the COBOL MOVE statement.

NIPS

The following statements may be used to change the values of both fixed and variable length items.

```
     MOVE operand1 [TO] operand2

                                          {SUBROUTINE  subroutine-name}
     MOVE operand1 [TO] operand2 USING    {TABLE       table-name     }
             EXIT label
```

In the second statement, label is the label of the NFL statement that will be executed from the error exit of the table or subroutine. Operand1 must be a fixed length item.

The following statement may be used to change the values of fixed length items.

```
     COMPUTE operand = arithmetic expression
```

Fixed length items are set to their null-values by

```
     DELETE FIELD operand
```

This same statement will actually delete a variable length item.

Variable length items may be inserted by

```
     ATTACH operand1 TO operand2
```

This statement appends the contents of operand1 to the variable length operand2. No provision is made for inserting fixed length items.

Operands may be constants or items from the file entry or the transaction

TDMS

The SET command with an optional WHERE conditional expression is used to change item values in both the MAINTAIN and UPDATE modes. A null value can be assigned to an item by a comtination of SET and FAILS as for example

SET AGE EQ FAILS WHERE EMP NAME EQ JONES

In MAINTAIN, the object of the SET command can be more complicated than in UPDATE. It may include arithmetic expressions containing items from both the current file entry and transaction.

UL/1

The statement for changing an item value is

$$\text{itemid} \left\{\begin{matrix}\text{REPLACE}\\ \text{R}\end{matrix}\right\} \text{WITH value}$$

if the nonnull value of a data file item is to be changed regardless of its current value in the data file. If the change is to take place only if the value of the item in the data file has a specified value called "filevalue", the statement is

$$\text{itemid} \left\{\begin{matrix}\text{REPLACE}\\ \text{R}\end{matrix}\right\} \text{filevalue WITH newvalue}$$

To change a value in a multiple valued item, the following statement is used

$$\text{itemid REPLACE ordinal-position WITH} \left\{\begin{matrix}\text{value}\\ \text{ordinal-position}\end{matrix}\right\}$$

Ordinal position is expressed as FIRST, LAST, SUBITEM n, or LAST BUT n.

For single valued items, only those that exist with null values can be inserted. In that case

$$\text{itemid} \left\{\begin{matrix}\text{INSERT}\\ \text{I}\end{matrix}\right\} \text{value}$$

changes the null value to a data value. If a value is to be inserted in a multiple valued item, the following statement may be used:

$$\text{itemid} \left\{\begin{matrix}\text{INSERT}\\ \text{I}\end{matrix}\right\} \text{value IN ordinal-position}$$

The statement for deleting a nonnull single-valued item (the only kind that can be deleted) using a "discrete" update is:

$$\text{itemid} \begin{Bmatrix}\text{DELETE}\\ \text{D}\end{Bmatrix}$$

For multiple valued items, the following statement must be used

$$\text{itemid} \begin{Bmatrix}\text{DELETE}\\ \text{D}\end{Bmatrix} \quad \text{value IN ordinal-position}$$

SC-1

Item level changes are made by first using a GET statement (see 5.3.3) to retrieve the group containing the item to be changed. This puts it in a buffer area. Once the group is in a buffer area, item value changes are made with SET statements (see 5.3.5.1). When all desired item changes have been made, the group is returned to the file with a

REPLACE group-name IN file-name

5.3.6 Transaction validation

When viewed across a number of systems, a variety of tools can be supplied to the user for checking the validity of transactions. The checking may be simple checks on item values to see that they are of the type of within the limits supplied by the user in the transaction definition. The user may also be allowed to specify logical relations that must hold within a transaction before it can be applied to the file. In other cases, the user is allowed to specify logical relations that must hold between the transaction and the file data before the transaction can be applied to the file.

GIS

In both UPDATE and MODIFY the conditional expressions used for the interrogation function (see 4.3) are available for the user to specify transaction validation. Transactions are submitted to the system as files so the validation that results from file definition (see 5.4.1) is also performed.

MARK IV

The user controls transaction validation by stating the conditions to be met on the Transaction Definition Form or the Processing and Record Selection Form. He may state minimum and maximum values for items, specify "pictures" that indicate the acceptable class of characters for each character position of an item, or ask for validation of items containing dates. If the system supplied character classes of A-Z, 1-9, only A-Z, or 1-9 and blank, etc., are not satisfactory, the user may define his own character class. If these checks are put on the Transaction Definition Form, the system will apply them to transactions as they are received. In addition, the user may specify similar checks in his transaction processing.

NIPS

The processing to be performed is implied from the transaction definition or is specified using NFL statements. An automatic check is made on all data defined as numeric to be sure it contains only + and - signs and decimal digits.

NFL statements that are particularly useful for data validation are the IF statement with the usual relational operators and the item PICTURE test. The MOVE statement can also be used for item validation because it provides for moving an item via the validation processing of a subroutine or table.

TDMS

MAINTAIN can perform its validation processing on data derived by the use of statements containing arithmetic expressions. These expressions may use item values from both the current transaction and the file entry.

UL/1

The REPLACE statement (see 5.3.5.3) allows the user to state what he believes the present value of a file item is in addition to supplying a new value. When this is done, the value given is compared with the actual file value and if they are not equal, no updating takes place.

The user can use the word ITEMIZE at the beginning of the Update Division to cause the printing of a listing of the old and new contents of each item in each entry changed during the update run.

SC-1

Each of the four types of transaction validation provided on the transaction definition form also exists as a statement that can be used in a transaction program. The error clause for each of these statements takes the following form:

```
    ERROR  [error-message-code]  [error-action-code] [SKIPTO label]
```

The validation statements are:

TYPECHECK transaction-item-name [MUST] [[n] type-code] ...
[error-clause] ;

VALUECHECK transaction-item-name expression

$\left[\begin{Bmatrix}\text{OR}\\ \text{TO}\end{Bmatrix}\text{ expression}\right]\ \ldots\ [\text{error-clause}]\ ;$

TOTAL CHECK transaction-item-name1 transaction-item-name2

[error-clause] ;

$\text{TEST}\ \begin{Bmatrix}\text{(Boolean-expression)}\\ \text{transaction-item-name relational-operator (expression)}\end{Bmatrix}$

[error-clause] ;

These statements can be combined with the GET and SET statements that set up a group in a buffer area and change item values in it. Items from the buffer are also acceptable variables for expressions in these validation statements.

These same transaction validation operations can be specified as part of transaction definition (see 5.2.1). When this is done no buffer items may be referenced and the SKIPTO option in the ERROR clause is not allowed. The user may set up a separate run on a batch of transactions to sequence check them on up to four items and to see if the batch checks to a predetermined count of transactions, item sum, or item value.

5.3.7 Transaction editing and transformation

Systems may provide extensive facilities for editing and transforming transactions before they are applied to the file. Sometimes editing consists only of supplying or truncating leading or trailing zeroes or blanks. Transformations the user may specify can include such things as coding or decoding transaction items and the application of computational algorithms to the transaction or to combinations of the transaction and file data.

GIS

Transactions are presented to the system as a file. File definition provides for editing through the use of pictures, masks, tables, and subroutines, and transformation through encoding and decoding tables and subroutines (see 3.2). These are automatically applied as new data enters (see 5.4.1). They may, however, be overridden by the IGNORE statement which has the following form:

$$\text{IGNORE filename} \left\{ \begin{array}{l} \text{EDIT/ENCODE} \\ \text{DECODE} \\ \text{ALL} \end{array} \right\} ;$$

The option chosen from within the braces is applied to all items in the file. The ALL option means that all editing, encoding and decoding specifications in the data definition will be ignored during updating.

Additional editing and transformation may be supplied by using conditional statements and arithmetic expressions used for interrogation (see Chapter 4).

MARK IV

When the Transaction Definition Form is used, transaction items may be added to or subtracted from file items. Numeric data in string form will be converted to the appropriate type and leading zeroes are supplied before the transaction items are applied to the file.

When using the Processing and Record Selection facility, table look-up transformations of both file and transaction item values can be specified. The tables used must be defined and processed in a dictionary maintenance run prior to the update run. Arithmetic expressions using items from the current file entry, and transaction or from consecutive file entries are also available for performing data transformations.

When the Processing and Record Selection facility is used in conjunction with the Transaction Definition Form, the user can specify arithmetic processing, item justification and some string manipulation which will take place before or instead of the automatic transaction processing.

NIPS/FFS

Tables may be used to code or decode data, and subroutines may be used to transform it. The tables and subroutines must be stored in a subroutine library prior to the update run. The subroutine library may be stored as part of the data file. Tables provide for validation by a simple correspondence between input values and the values to be used by the transaction program. Subroutines can be written in any OS/360 supported programming language. They may not perform input and output operations and the subroutine output must contain a specified code indicating whether execution was successful or not.

Edit masks are provided at file definition time and in output processing. For transaction editing, the user writes his own using mainly IF and MOVE statements as necessary.

TDMS

When using MAINTAIN, data transformations can be obtained through statements containing arithmetic expressions that use item values from both the current file entry and the current transaction as operands.

UL/1

The update language allows the specification of computational procedures that can be used to perform computations on numeric items already in the old file entry before a new file entry is stored.

SC-1

Data items may be justified in any one of four ways with

$$\text{JUSTIFY transaction-item-name} \begin{Bmatrix} \text{RJZ} \\ \text{RJB} \\ \text{LJZ} \\ \text{LJB} \end{Bmatrix} \;;$$

where RJ stands for right justify, LJ for left justify, Z for zero fill, and B for blank fill.

Transformations can be performed by using arithmetic expressions in the SET statement (see 5.3.5.1).

5.3.8 Other user specified features

Many other features may be provided as a part of the update function. Only one is discussed.

SC-1

The SURVEY statement is similar to the TEST statement, but instead of being used to check on the validity of transaction data it is used to check data in the data file. The statement syntax is:

SURVEY (boolean-expression) [error-message-code] [SKIPTO label] ;

If the boolean expression is true, the error message is printed. No SC-1 error action may be specified, but the same effect may be obtained by the use of the optional SKIPTO phrase.

5.4 Auxiliary update functions

Update processing could be initiated by the system on the basis of the time of the day, the date or data conditions. In the system analyzed, however, it is initiated by the user submitting a job in the batch mode, initiating a job from a terminal or submitting a transaction from a terminal. Once the execution of an update transaction program is initiated, certain processes may accompany its execution that are not specifically asked for in the transaction program. Some of these processes may be initiated to preserve conditions specified by the user at file definition time such as checks to preserve item data types or checks to see that the user has complied with a system requirement like submitting transactions in file entry identifier sequence. Others such as writing run histories, are provided to make it possible to verify what actions affecting the data file took place during the run. Some actions that might be auxiliary update functions such as the automatic creation of audit trails are discussed as part of data administration (see 8.2). If systems have such facilities they are mentioned here to indicate that they take place during updating.

5.4.1 Maintenance of item attributes

The maintenance of file item attributes so that they conform to the file definition usually takes the form of automatic conversion, if necessary, to file item type before it is entered into the file. The treatment of nonconforming item lengths, and the application of file definition criteria varies with the system.

GIS

Automatic conversion between packed decimal and right-justified EBCDIC is provided for numeric representations. Values entered into the file must agree with the file definition type, length and editing criteria and are subject to encoding or decoding by subroutines or tables provided in file definition.

MARK IV

Users provide numeric items as decimal numbers, but all OS/360 data forms are used internally and automatic conversion is supplied as needed. File definition includes the specification of the length of numeric items and the length or acceptable range of lengths of string items. All values entered into the file must agree with these lengths.

NIPS/FFS

Before an item is entered into a file it is checked for conformance with the file definition item type. If necessary, acceptable conversions are automatically made between the item types alphanumeric, binary, coordinate and decimal. If lengths do not agree, alphanumeric items are padded with blanks or truncated on the right, and numeric items are padded with zeroes or truncated on the left before being entered into the file.

TDMS

All new values for an item are subjected to validation checks specified in the DEFINE operation. This is true for new data initially entered through GENERATE as well as for MAINTAIN and UPDATE.

UL/1

Item values are checked to see that they match the file definition on type, length and validation criteria. Attempts to DELETE a null valued file item or INSERT an item that exists in the file with a nonnull value cause an error. This checking is performed after all updating changes have been made. If the updated entry is found invalid, the old file entry is retained. Rejected transactions are listed.

SC-1

The execution of the SET statement (see 5.3.5.1) and transfer of data from the buffer area to the file causes item type conversions to take place. Checking is not a system facility, but is user programmed using transaction validation statements (see 5.2.1 and 5.3.6).

5.4.2 Maintenance of file storage

During the updating process file storage limits may be exceeded. The enforcement of these limits is often left to the operating system, but it and other storage maintenance functions are sometimes performed by the data management system.

GIS

Performed by OS/360

MARK IV

When groups or entries are being inserted, a check is made to be sure that the physical limits of the file are not exceeded. If they are, the transaction is rejected.

NIPS/FFS

Performed by OS/360

TDMS

During updating, the update storage requirements are checked against available storage. If space is not available, the updating is terminated. MAINTAIN also includes a copy and clean up function that regenerates spare storage after an update has taken place.

UL/1

During updating each entry is checked to be sure the over-all length of the updated entry does not exceed 7000 bytes.

SC-1

During updating, blocks of storage are made available within the data sets assigned to files. If the storage needed exceeds that assigned, the run is closed. A utility program can then be used to increase the amount of physical storage assigned, and updating continued in a new run.

5.4.3 Ordering of transactions and files

Particularly in systems that use sequential files, update processing can be speeded up by processing the transactions in the same sequence in which the affected entries are stored in the data base. The ordering of transactions may be a user responsibility. Sometimes the transactions are accepted and processed in any order, and sometimes the system accepts them in any order and if necessary, sorts them into file sequence before processing.

GIS

Transactions submitted in the batch mode that are to be entered into a sequential file are checked to be sure they are in sequence on the file entry identifier. If they are not, the transaction is rejected and a message produced.

MARK IV

Transactions applied to a sequential file are checked to be sure they are in sequence on file entry identifier. If they are not, the transaction is rejected and a message produced.

NIPS/FFS

In the batch mode, transactions are checked to see if they are in sort on file entry identifier. If they are not, they are sorted before they are processed. From terminals, transactions are accepted and processed in any order.

TDMS

MAINTAIN, which operates in the batch mode, converts the transactions to a TDMS structured file before the execution of the update.

With UPDATE, which is used from the terminals, transactions are accepted and processed in any order.

UL/1

During a "discrete update" transactions are checked to see if they are in file sequence. If not, they are sorted. If they are, the sort is bypassed.

SC-1

The setup run that enters transactions into the SC-1 transaction file requires that the user specify the sorting of transactions that will put them in the same file entry identifier sequence as the file they are to be used with. During updating the sequence of the data base file is checked. If it is not in sequence, the run is terminated.

5.4.4 Maintenance of system integrity

During update it is possible to provide facilities that are not the result of user specification. The actions taken to prevent interference from the execution of other programs using the same files are discussed under operating environment concurrency (see 10.2.1.2).

GIS

A run history that indicates recognized errors is produced automatically.

When UPDATE is used, an audit trail consisting of a before and after listing of changed entries is made automatically.

MARK IV

A check is made to prevent entries with duplicate entry identifiers being put in the file during updating.

NIPS/FFS

A run history that indicates recognized error conditions is produced automatically.

TDMS

Backup files and audit trails are not defined features, but MAINTAIN automatically produces a new file.

UL/1

Rejected transactions are listed automatically along with an indication of the errors causing the rejection.

Three general types of errors, translation, comparison and validation are checked for. Translation time errors are things like incorrect language forms or data types. Comparison time errors include things such as a transaction that is received for an entry not in the file or an attempt to insert an entry with an identifier that is the same as one already in the file. Validation errors occur when the final form of the updated entry fails to meet the criteria set up in the data definition for the file.

SC-1

Backup files are written by the system, but they cannot be substituted for data files. The user must write a program that will recreate the data file using the backup file as input.

6. CREATION FUNCTIONS

The creation of the initial instance of a file or data base is the process of making known to the data base management system a set of entries on which it can perform other functions. This may mean nothing more than entering a data definition (see Chapter 3) for a file which already exists in machine processable form. It may imply the conversion of an existing file into a form acceptable to the system. This conversion process may be performed by a self-contained function which is part of the system, or it may need to be programmed in a conventional sense using programming facilities (see Chapter 7).

In the case that a self-contained function is provided, it may be essentially a special use of an update function (see Chapter 5) or it may be a facility provided specifically for the purpose of file creation.

Data definition of the master file is always considered an inherent part of the creation process, whether the file to be identified to the system as a master file is converted from some input file or whether the existing stored file is to be made known to the system by the entry of a data definition. The definition may be of an entire data base or some portion of a data base. In this latter sense, it may adjoin a file or part of a file to an existing data base.

Data definition facilities for the input file (or the system required format), and specifications of the validation conditions which entries stored in the master file must satisfy, are also considered part of the creation function.

The reports which the system generates as part of the creation process may be quite extensive and needed by subsequent users of the system. Control by the user over the media type for the initial instance of the master file is usually an operating system consideration (see Chapter 10) but is a feature of the creation function in the sense that the user may have to decide on storage space requirements as well as on media type (see Chapter 8).

Excluded from this discussion are such topics as:

- problems of mobilizing human collection of data,
- problems of designing adequate inputs,
- re-creation of a file by loading a recovery dump and
- creation of the stored data definition as a file creation problem (see 3.8).

The concept of creation as used here includes the re-creation of a file. Re-creation is a variant of that form of creation which converts an existing file into the desired structure of the file or data base.

Creation functions are discussed in this chapter only if they are provided by the system as self-contained functions. The host language systems provide for data structure definition, storage structure definition, allocation of media space, provision of input and population of the master file. However, these actions are not organized into an independent creation function and the population of the file is performed through the use of programming facilities (see Chapter 7) or by populating a null file using the update functions.

GIS

Creation requires the data structure definition and the selection of access method for the master file. The defined file may be populated by means of a set of procedural statements (similar to those used in update) in a CREATE subprocedure. An already existing file may be made known to the system if it can be described in the data structure definition.

MARK IV

Creation requires the data structure definition and the selection of access method for the master file. The defined file may be populated by the update function or may already exist in an acceptable format and require only that it be made known to the system.

NIPS/FFS

Creation requires the data structure definition and the selection of access method for the master file. The defined file is populated by means of the update function or it may already exist in an acceptable format and require only that it be made known to the system.

TDMS

Creation requires the data structure definition for the master file. The defined file must be populated and this is normally done by a special creation function called GENERATE or by means of the update function.

UL/1

Creation requires the data structure definition for the master file. The defined file must be populated and this is done by a special function called ESTABLISHMENT or by use of the update function. If the master file is not to be updated, any file capable of definition in the COBOL data description can be made known to the system.

6.1 Creation action cycle

In the creation or re-creation of a file there are seven basic steps:

- definition of data structure for the file and identification of the part of the data base being created (see Chapter 3),
- definition of storage structure for the file (see 9.4),
- definition of data structure for the input source file (see 6.2),
- definition of storage structure for the input source file (see 6.2),
- allocation of media space for the file (see 6.3),
- provision of data on an input source file (see 6.4) and
- population of the file (see 6.5).

The definition of the data structure for the input source file makes use of all or part of the facilities provided for the data structure definition of the master file (see Chapter 3) or is accommodated by a special definitional capability to accommodate transactions like those used in update functions (see Chapter 5).

The definition of storage structure for the file is either provided automatically or specified in part by the user (see 9.4). However, a choice of access methods to be used on the input data file is usually provided to the user.

The allocation of media space for the file is the reservation of storage space for files (and possibly their related schema). This allocation function is achieved by a statement of data definition which provides parameters to the operating system space allocation facilities. The operating system allocates space on the physical storage medium and keeps track of space usage.

The provision of data on an input file is the step which prepares data for presentation to the population step. The system may accept a foreign file created outside the system or a file constructed by use of the interrogation functions from files existing in the system. The input file may be not only a physical file but also a stream of transactions which can be viewed as the result of a procedure which provides entries without the user being aware of an intermediate file. Transactions presented from a terminal in an interactive or conversational mode are examples of such a procedure (see 10.3.2).

The population of the file can be achieved by the simple means of defining an existing file in such a way that it becomes acceptable to the system. This in effect telescopes the provision and population steps of the cycle. Otherwise the population of the file is in the mapping of data from the input file to the data base and is usually performed as a stand alone task. This task may be accompanied by built-in error detection procedures or may permit the use of the interrogation function facilities to provide reporting on the progress of the population step. This step can also be accompanied by special restart and recovery procedures.

The steps of the creation action cycle are normally partitioned in some way since the entire cycle is an extensive task. The degree to which these steps can be combined or done in parallel is significant in understanding the use of the system (see 10.2.3). The actions of monitoring the progress of the provision and population steps are especially affected by the ability to use interrogation functions to combine data extraction with reporting and to use interrogation functions during creation or update.

Re-creation of the data base involves the seven steps with some variation. Revision of the data structure of the file is discussed with data structure definition (see 3.9). Definition of the data structure for the input file and provision of data on the input file can be subsumed in the original creation of the file being restructured if the system provides a stored data definition and the use of a system file as input to the creation process.

GIS

The definition of data structure and access methods for the master file are performed in a single step of data description. The definition of the data structure for the input data file is performed in a data description task. This task involves the definition of a sequential file of the "mulrec" (multiple record) form. Existing system files and "hold" files may also be used as inputs to the creation process. Allocation of media space is defined in data management statements using the operating system facilities. Provision of data is in the form of a file or stream of data from a terminal. The population of the file is described in a procedural task which can include conditional and listing statements and which is compiled in a separate step. Each of the steps in the cycle can be done separately or may be combined into as few as two steps:

- definition of data structure for both master file and input data file and allocation of media space and
- definition and execution of provision of input data and population. This latter step can be eliminated if an existing file can be defined as is and is made known to the system.

MARK IV

The definition of data structure and control over the access method for the master file are performed in a single step of file definition. The definition of the data structure for the input data file is combined with description of population or provision actions by defining a transaction for update or interrogation. Allocation of media space is defined in data management statements using the operating system facilities. Provision of data is always on a file. Population of the master file is by use of the update function and is accompanied by some monitoring. These steps can all be done separately, if desired, or done in as few as two separate groups:

- definition of master file, input data file, allocation of media space and definition of transaction programs to provide input data and provision action and
- definition of the population task and its actual performance. This latter step can be eliminated if an existing file can be defined as is and is made known to the system.

NIPS/FFS

Definition of file data structure is done in the data definition task. Definition of the input file data structure is combined with definition of the provision or population transaction programs for use in update. Allocation of media space is defined using the operating system facilities. Provision of input data may be from a file extracted from existing files, a separately prepared file or a stream of transactions from a terminal. Population of the master file is by means of update transactions and includes some automatic monitoring. These steps may be done separately or combined in as few as two steps:

- definition of master file data structure and access methods and allocation of media space for the master file and

- definition of the population action and the performance of the action itself. This latter step can be eliminated if an existing file can be defined as is and is made known to the system.

TDMS

Definition of the structure of the file is automatically accompanied by internal definition of storage structure. Definition of the input file is an internal system specified function, but utility programs are provided to ease the conversion of input files in foreign formats. Allocation of media space is done in conjunction with the establishment of the system in its operating environment. Provision of input data and population of the master file are combined and the input data may be provided as a file or a data stream from a terminal.

UL/1

Definition of the data structure of the master file may be done separately but requires the provision of at least one entry and causes allocation and population with that single entry to take place in a single step. Definition of the input data structure is either provided by the system or by a definition which makes an existing file acceptable for provision of input. Allocation of media space is done automatically with the definition of the master file. Provision of data can be done by definition of a file to be produced by the interrogation function, by making the description of a foreign file (whose structure can be encompassed in a COBOL data definition) known to the system, or by providing input in the system provided format from a file or terminal. Population of the master file can be combined with master file definition or done in a separate step and is supported by monitoring capabilities.

6.2 Definition of data and storage structure of input files

Input data can be provided to the system's creation or update functions in a system specified format or in other user defined formats. These user defined formats can be described by the system data definition facilities, some subset of those facilities, or in some external language. The definition can make explicit a selection of access method for the input data or restrict it, as is common, to sequential access.

Input data can also be extracted from existing files and put into files acceptable as input to the creation or update functions. The definition of such files is either implicit in a system defined format or can be defined in the specification of the interrogation used to extract the file data.

Internal storage of the definition of data and storage structure for these input files is discussed in the data definition functions (see 3.10).

GIS

Input data may take the form of a system file or of a multiple record type file which is a sequential file of the same data structure class as system files and expressly designed for creation and update functions (see 5.2.1). In addition a hold file may be extracted from existing files using the interrogation functions for use as input data. Definition of the hold files is done implicitly in the interrogation processes giving rise to them. They are also sequential files of the same data structure class as system files (see 4.12).

MARK IV

Input data must be provided in the form of a sequential file with fixed length entries. Definition of these files is accomplished by the definition of the transaction program used to populate the file through the update function (see 5.2 and 5.3). Files in the appropriate sequential fixed format can be generated from existing system files by the interrogation function's data selection facility (see 4.6.1 and 4.13).

NIPS/FFS

Input data must be provided in the form of a file of transactions suitable for the update function. The definition of the data structure of these files is accomplished by the definition of the transaction program used to populate the file in the update function (see 5.2 and 5.3). Files in the appropriate format can be generated from existing system files by the interrogation function's data selection facility (see 4.6.1).

TDMS

A system defined input format for file creation is provided. Input to the GENERATE operation is a continuous string of characters. A definite format and sequence of the fields within the string is required in order to identify the relationships of each character. This format is of the form

item-number) b data value

where the item-number is assigned to uniquely identify the item in the definition of the master file. As an example,

1) 37562 2) "JONES P."

3) 42074 4) "123 MAIN STREET"

When data occur in a repeating group, all associated values (with their item numbers) are preceded by the group number. GENERATE requires that all input data be grouped by entry and that each entry be terminated by the symbol specified in the DEFINE operation. Within the entry all data values for level-zero items must be entered before any group data but may be entered in any order. The omission of an item number and value indicates that the value for that item is null. Group data are entered following the values for the level-zero items. All the data values associated with one level-zero group are entered before entering any data belonging to a different level-zero group. The order in which the groups of the same level appear is not significant. The organization of data within a higher-numbered level-group follows the same pattern as data for level-zero items and groups. That is, the data values for the items must be entered before the group data. Also, all the data associated with one group must be entered before entering data belonging to another group of the same level. Since virtually no existing data files exist in the required system defined format, generalized programs have been developed to ease the conversion from any existing format.

UL/1

The input is provided in a system defined format or in a COBOL format. The format of the data comprising an entry is the same whether it is input in the Establishment Division or in the Update Division. Each data item value must be preceded by one of the two item identifiers, the item number of the item name. Alphanumeric items containing spaces or reserved characters must be enclosed in the system's string delimiters < and >. An example of a simple input record is:

#1 37562 #2 <JONES P.> #3 42074

#4 <123 MAIN STREET> #5 HOMETOWN #6 CA*

If the file contains multiple valued items, or repeating groups then parentheses are used to designate group membership. An asterisk indicates the end of the entry. A section of the Establishment Division called the Layout Section is used to facilitate the establishment of files which already exist in machine readable form. In the Layout Section the user may define the layout of data items within a file of fixed-field fixed-length entries. Blocking factors are handled automatically by the system for such files. The COBOL record description may be used to make files known to the system for purposes of interrogation by the system to generate files acceptable as input data (see 4.6.1).

6.3 Allocation of media space

The definition of structure may or may not be separated into data structure definition and storage structure definition. However, the task of space allocation is separated internally and manifested by the presence of diagnostic messages from the operating system indicating the presence or absence of adequate file space in the systems.

The space requirement can be estimated either by the system or by the data administrator from the data structure definition augmented by data on the number of entry instances and expected rate of increase resulting from update. Limitations on the media types permitted for storage of the file can be inferred from the means required to access data for updating and interrogation and the response constraints on these access methods.

Space requirement considerations are somewhat different for input and file requirements and most of the self-contained systems make some simplifying assumptions on the space requirements for input, for example, that it be provided serially and processable at the time of file population.

Systems operating under OS/360 have available allocation facilities for data sets which include both input and file. In general the user provides the specification of the space required, the medium and any specification required to partition the medium. This allocation will be recorded by the system so that an attempt during the population action to exceed the allocated space will result in a message and termination of the process. OS/360 provides for the specification of overflow areas and for interchangeability of disk and tape in the case of sequential access methods. ADEPT 50 under which TDMS operates and TDOS under which UL/1 operates offer comparable facilities.

GIS

Use of overflow areas can be specified by the user and the system reports on use of space made during population. Space reservation is accomplished through data management statements reflecting user estimates of space which he requires and declares in file definition (see 9.4). Diagnostic messages are provided during population on inconsistencies between access method and media used as well as on entry overflow.

MARK IV

Space reservation is accomplished through data management statements reflecting user estimates of space which he requires and declares in file definition (see 9.4). Reports are made during population on media use inconsistent with the access method, on file names which have not been catalogued and on entry overflow or entry data space which exceeds track capacity or physical block (segment) capacity.

NIPS/FFS

Space reservation is accomplished through data management statements reflecting user estimates of space which he requires and declares in file definition (see 9.4). Diagnostics are provided during population on the storage of control information which exceeds the maximum permitted.

TDMS

The user must provide an initial estimate of file size. Space needs encountered during population of the file are filled by the operating system based upon this estimate. Need for additional space will be filled up to the capacity of the volume specified.

UL/1

The user must estimate the number of disc tracks required for temporary working space. He must do this also for the stored data definition (which includes validation criteria, code table and permanent computation procedures) and for the master file, if these are to be stored on disc. Assignment of media type (including which kind of disc) is done by TDOS operating system control statements.

IDS

A utility program, called QUTU, is provided to allocate space for the data base. The user supplies page and optional page range data consistent with the definition of the storage structure (see 9.4). The utility establishes page headers and initializes the inventory of pages.

IMS

A utility program for data base description generation, called DBDGEN, is provided to allocate space for the data base. This utility is constructed by the user from macro instructions in a library called IMS2. MACLIB. Space can be allocated in terms of blocks, tracks or cylinders as well as from logical record size and blocking factors. The device type may be indicated. User specification of data definition and access method (see 9.4) are taken into account.

SC-1

A utility program, called Data Location Definition, is provided. It accepts information from the data structure definition to establish the physical location of logical segments of a primary file. User specification of the blocking and access methods is used in this program (see 9.4). Parameters are provided to indicate that a first generation of a file is being allocated, the device type being used, the identifier of the part of the data base being created, block size of records, and proportion of space to be left for future expansion.

6.4 Provision of the input data file

The input data file may have to be specially prepared in a system required format. Alternatively the system may provide facilities defining the data and storage structure of data already in machine readable form. In this latter case the system may process the data in its existing form or may perform some kind of transformation. In general this provision activity contains the following steps:

- data is read and selected,
- transformations are made within entry instances,
- inter-entry calculations are made on item instances and
- reporting is done on the result of inter-entry calculation to provide control over completeness.

Two additional aspects of the provision activity where machine readable data already exists are aided by the interrogation functions of these systems (see Chapter 4). First, data may be reviewed to determine the size of the file or data base to be created and second to establish need for editing, restructuring and other alteration of data. Restructuring may be required to provide a more convenient input data file to the population activity when a change of master file structure is radical enough to cause problems in relating the existing instances to the new structure. Editing can include decoding and encoding of coded data, addition of data from other files and the like, where this capability is not available during the population step (see 6.5).

GIS

A single input data file with a sequential or indexed sequential storage structure can be processed provided these entries contain an item in the same position in each group whose value identifies the group or provided that the group can be identified by use of count fields. This file can be prepared from existing machine readable data in a separate step for the purpose of establishing master file size and editing requirements by means of interrogation facilities (see 4.13). This step can be combined with file population in a single step where there is no need for review for size or editing.

MARK IV

A single input data file with a sequential storage structure is required for the population step. The identification of the entry is provided in the transaction definition used in the population step (see 5.2.1). Such a file can be prepared through the use of the interrogation function as a separate step for the purpose of establishing master file size and editing requirements (see 4.13) from existing machine readable data.

NIPS/FFS

A single input data file with a sequential or indexed sequential storage structure is required for the population step. (The entries can be provided from a terminal using the update function to populate the file.) The identification of the entry is provided in the transaction definition used in the population step (see 5.2). Such a file can be prepared through the use of the interrogation function as a separate step for the purpose of establishing master file size and editing requirements (see 4.13) from existing machine readable data which is already in the system. When the interrogation facility called RASP is used to prepare input source data, a utility program called UTQRTQDF is provided to convert RASP output to a sequential tape file which contains the extracted data and the file definition internal tables as well as prestored transaction programs associated with the parent file. The RASP output is restricted to data from a single file and may not contain duplicate entries resulting from a single interrogation. Also the RASP interrogation may not contain multiple IF conditions (see 4.), multiple SELECT statements (see 4.2) or sorts of the extracted data (see 4.6.4).

TDMS

Input source data must be provided in a special system format (see 6.2). Programming aids are provided to convert existing machine readable files into this special system format. Analysis of such data not in system format would also be programmed outside the system.

UL/1

A single input data file is used to populate a file. This file may be in the special system format for this purpose (see 6.2), or it may be a sequential file which can be described by a COBOL data definition. Files describable in COBOL can be made known to the system for purposes of interrogation so that they can serve either to telescope the activities of provision and population or to furnish the data for a file. This file would be produced by the interrogation function as a separate step for the purpose of establishing master file size and editing requirements (see 4.13).

6.5 Population of the file

The population of the master file or data base can be achieved by the simple means of defining an existing file in such a way that it becomes acceptable to the system as a file or data base. Otherwise the population of the file is the mapping of data from an input file to the file or data base. This mapping may be a simple copy of entries from the input file into the master file or may require a transformation of items or the regrouping of items into different groups from the input to the master file. The method used for population may be the update function or a function specialized for the purpose of creation. The process of populating the file is normally accompanied by facilities to validate the acceptability of the data by testing item values and the consistencies of items within groups or entries as well as the consistency of entries within the file. Also facilities can be provided to insure that all the data expected to be processed from the input file was put in the master file. The processing in the population action involves the following steps:

- reading and intra-entry validation of entries in the input file,
- testing for duplicate entries or entries invalidated by inter-entry tests on the input file,
- collection of counts or values of source data for inter-entry validation or control totals,
- transformation of input source entries or items to the format of the master file,
- collection of counts or values of master file data for inter-entry validation or control totals,
- logging or dumping of master file contents to provide recovery in case of hardware failure during the process and
- comparison of inter-entry data collected with control information to check completeness of the process.

GIS

The population action is specified in a CREATE subprocedure with the form:

```
CREATE master-file FROM source-file
create-statements
END PROCEDURE
```

Transformations are done by means of the create-statements and are subject to the same limitations as statements used in the update functions (see 5.3.7). Validation of items is automatically provided on the basis of item attributes (see 2.1 and 3.2). Validation of items, groups and entries in the input data can be provided by procedural statements which make use of arithmetic and other operation on item values and the values of temporary variables. Entries and groups can be listed if they fail validation tests. Logging or dumping of a partially populated master file is not provided. A report or counts and control totals of both input data and file data processed can be provided by means of procedural statements and the use of temporary variables.

MARK IV

The population action is specified by use of the update functions (see 5.3). A special transaction code for creation provides for moving new entries from the input file into the master file. Transformations and validation are described in the transaction program definition (see 5.3.7 and 5.3.6). Validation of items is specified by the user in the transaction definition in terms of item attributes (see 2.1 and 3.2). Validation of groups and entries in the input data can be provided by the use of arithmetic operations on item values and the values of temporary variables. Logging or dumping of a partially populated master file is not provided. A report on counts and control totals of both input source data and master file data processed can be provided by means of the transaction definition and the use of temporary variables. Automatic counts of entries processed from the input file and entries added to the master file are provided.

NIPS/FFS

The population action is specified by use of the update functions (see 5.3). Transformations and validation are described in the transaction program definition (see 5.3.7 and 5.3.6). Validation of items is specified by the user in the transaction definition in terms of item attributes (see 2.1 and 3.2). Validation of groups and entries in the input data can be provided by procedural statements which make use of arithmetic operators on item values and the values of temporary variables. Logging or dumping of a partially populated master file is not provided. A report on counts and control totals of both input data and master file data processed can be provided by means of the transaction program and the use of temporary variables. Utilities are available, called UTBLDISM and UTBLDSAM, which can convert, respectively a tape sequential file to a 2314 ISAM file and a 2314 ISAM file to a tape sequential file. These utilities can be used to populate a file on a different medium or with a different access method from an existing file.

TDMS

The population action is specified by use of the GENERATE function or the update functions (see 5.3). Transformations and validation are described in the transaction program definition when update functions are used for creation (see 5.3.7 and 5.3.6). Transformations are not provided at the time of the population process. Validation of items is specified by the user in terms of item attributes (see 2.1 and 3.2). Validation of groups and entries depends entirely upon validation of items. Logging or dumping of a partially populated master file is not provided but population can be suspended and restarted. A report of counts of data placed in the data base is provided.

UL/1

The population action is specified in the ESTABLISHMENT division in the form

ESTABLISH file-name

An existing file, with a data and storage structure which can be described by a COBOL data definition, can be made a file in the system. However, this file can be used only for interrogation and cannot be updated. Validation of items is automatically provided on the basis of item attributes (see 2.1 and 3.2). Validation of items, groups and entries in the input source data can be provided by procedural statements in the transaction definition which make use of arithmetic and other operators on item values and the values of temporary variables. Entries and groups can be listed if they fail validation tests. Logging or dumping of a partially populated master file is not provided. A report on counts of entries in the master file is provided automatically.

6.5.1 Entry, group and item validation

The validation of entries or groups of items usually implies some function on already validated item values which are compared with other data items within the group or entry or the results of computations on such data or constants. The validation of items is specified in the item definition (see 2.1 and 3.2) or in the transaction definition where population is by means of update functions (see 5.3.6). Validation of groups or entries may also be provided between groups or entries to insure that identifiers of groups or entries are unique, that sequencers are in order in the input data and the master file, or that the overall size of entries is within some maximum.

GIS

When population consists of making an existing file known to the sytem, no validation processing is provided. In addition to item validation sequencer items may be required to be unique (see 2.2). The creation statements may be used to provide conditions to test for consistency among the items of a group or entry as well as to test the sequence of sequencer items between group or entry instances. Operators and temporary variables to provide counts, sums, averages and simple arithmetic functions are available for inclusion in the creation statements processed during population. Groups or entries which fail validation tests may be listed and included or excluded from the file.

MARK IV

When population consists of making an existing file known to the system, no validation processing is provided. When population is done by means of the update functions, the validation is that provided by the transaction programs used for update (see 5.3.6). These transaction programs may make use of operators and temporary variables to provide counts, sums and other simple arithmetic functions and conditional tests to assist in validation. Duplicate identifiers cause the rejection of the entry or group after the first occurrence of the identifier. Groups or entries which fail validation may be placed on a transaction reject file for later listing and may be included or excluded from the file.

NIPS/FFS

When population consists of making an existing file known to the system, no validation processing is provided. When population is done by means of the update functions, the validation is that provided by the transaction programs used for update (see 5.3.6). These transaction programs may make use of subroutines using operators and temporary variables to provide counts, sums and other functions and conditional tests to assist in validation. Groups or entries which fail validation are placed in a run history file automatically for later printing and they may be included or excluded from the master file being created.

TDMS

When population consists of the use of the GENERATE function, validation is limited to item validation (see 2.1 and 3.2). Items which fail validation are returned for interactive correction or the entire entry is rejected for batch correction.

UL/1

When population consists of making an existing file known to the system, no validation processing is provided. When population is done by means of the ESTABLISHMENT DIVISION or the update functions, validation is defined in the form of selection criteria (see 4.3) and arithmetic and statistical functions (see 4.7.4). A check is made for duplicate entries and only the first one arriving is allowed in the master file. Diagnostics reporting validation failures and entries having duplicate identifiers are printed. The length of an entry is restricted to 7000 bytes.

6.5.2 Transformations on data

During the activity of populating the file it may be necessary to transform the values of item instances, the order of items within a group, the order of groups or entries in the input data file, or to bring together two or more input files into a single master file. Within a group or entry, transformations can be provided which change the order of items or drop items, derive new items from old by arithmetic or other functions and encode or decode items by table substitution or special sub-programs. Transformations can be provided which change the order of entries or groups from the order which they have in the input data to some other order in the file being created. Other transformations may serve to create new items from counts or sums accumulated over a collection of groups or entries.

These transformations may be specified at the time of file definition, by procedural statements supplied to the population activity as transaction programs or the like, or at the time of input data provision (see 6.4).

GIS

The order of items within groups or entries and the encoding of item values is specified in data definition (see 2.1 and 3.2) and mapping from input data to file is automatic. Following the automatic mapping, items may be subjected to arbitrary arithmetic and logical transformation with the same operators used for validation. Sorting instances of groups or entries in the input data is required. Such sorting is done in the input provision activity. Population is done from a single input file so that merging of separate streams of input data would be done in provision of input or by entering other streams in update transactions.

MARK IV

The order of items within groups or entries is specified in data definition (see 2.1 and 3.2) and mapping from input data to file is automatic. Items may be derived by functions of arithmetic operators on item values, constants and the contents of temporary variables, and encoded or decoded by means of tables as specified in the transaction program associated with the population action (see 5.3.7). Sorting instances of groups or entries in the input is required and is done in the provision activity. Population is done from a single input file so that merging of separate streams of input data would be done in provision of input or by entering other streams in later update transactions.

NIPS/FFS

The order of items within groups or entries and the encoding of item values by table or special routine is specified in data definition (see 2.1 and 3.2) and mapping from transaction to file can be automatic. Items may be derived by functions specified in subroutines written in any 360 procedural language or associated with the transaction programs used in the population action (see 5.3). Sorting instances of groups or entries in the input data is done automatically. Population is done from a single input file so that merging of separate streams of input data would be done in provision of input or by entering other streams in later update transactions.

TDMS

The order of items within groups or entries is specified by the data definition of the file (see 2.1 and 3.2) but the input data is not required to be provided in this order and mapping is done automatically. No transformations to derive data item values are permitted in GENERATE although arithmetic functions are available in the update function (see 5.3.7). Sorting of instances of entries is unnecessary. Instances of repeating groups within entries need not be sorted but must be provided within the entry at the time of population. Any reordering of data required to provide entries must be done at the time of input provision. Population may be done from one or more input streams.

UL/1

The order of items within groups or entries and tables for encoding data are specified by the data definition (see 2.1 and 3.2). The input data may be provided in the defined order or not in which case mapping from the input file to the file is automatic. Items may be derived by functions of arithmetic operators on item values (see 4.7.4) within a group or entry as specified in the procedural statements associated with the population activity. When population is by use of either the update function or the ESTABLISHMENT division, sorting is done automatically. Population is done from a single input file so that merging of separate streams of input data would be done in provision of input or by entering other streams in update transactions.

6.6 Monitoring creation functions

Monitoring is the reporting of data reflecting the errors encountered in the various steps of the creation cycle and the statistics indicating the size of and resources used by files. In the course of the population activity, errors encountered by the validation criteria and data unacceptable to the input/output facilities of the operating system are normally reported. Correction of such invalid data can be on-line or batched. At the end of the population step, statistics indicating the amount of file storage used, counts of input file transactions and file entries created can be reported. The interrogation facilities may be available during the population activity to provide any reporting required by the user.

In the provision step in the creation cycle, validation errors and statistics on the size of the input file prepared may be reported. A separate analysis made of files ready to be used as input to the population function may be provided.

In the other steps of the creation cycle all systems provide diagnostic messages on violation of language syntax in data definition, transaction specification and storage allocation.

GIS

Errors detected by the system or by user procedures in the CREATE sub-procedure are reported. The user may select a code to control the severity level of errors reported. These errors include I/O errors, failures of the data to pass validation tests, arithmetic operation errors and overflow, groups, entries or files too large for allocated storage and missing hierarchical levels in data entries. Optionally the entries created can be listed as well as the transactions of the input file. After the population activity an automatic report is provided on the number of input transactions encountered and file entries inserted. For indexed sequential files, counts are reported for file entries in the primary file area, and entries placed in the overflow area. Counts of overflow cylinders filled, unused tracks still available in the overflow area, accesses made to the overflow area, levels of index used and tracks needed to accommodate the high level indexes are automatically reported. The interrogation capabilities cannot be used during population of the file. If provision of input is from existing file data as in re-creation or from system useable files, the interrogation capabilities are available to monitor the activity but no special analysis reporting is provided.

MARK IV

Errors detected by the system or by user specification of transactions in the update function used to create files are reported. Entries rejected by the validation criteria are placed on a reject file. On the output report provided to the user and printed from the reject file are also I/O errors, arithmetic operand conversion errors, groups, entries or files too large for allocated storage, entries or groups having duplicate identifiers and transaction sequencers out of order. Since the full capability of the interrogation function is available during update, any additional reporting required by the user can be requested. At the end of the population step there is an automatic report on the number of input transactions encountered and file entries inserted. For indexed sequential files counts are reported for file entries in the primary file area, and entries placed in the overflow area. Counts of overflow cylinders filled, unused tracks still available in the overflow area, and accesses made to the overflow area are also reported. If provision of input is from existing file data as in re-creation or from system useable files, the interrogation capabilities are available to monitor this activity but no special analysis reporting is provided.

NIPS/FFS

Errors detected by the system or by the user defined transaction programs in the update function which is used to create files are reported. These errors include, in addition to failures of data to pass validation tests, I/O errors, non-numeric operands in arithmetic functions and duplicate identifiers. Transaction programs used to create the file can also be used to select data and place it on another file (see 5.2.2). This file can be used to generate reports on the effects of the population step. After the population activity an automatic report is provided by the operating system on the number of input transactions encountered and file entries inserted. For indexed sequential files counts are reported for file entries in the primary file area, entries placed in the overflow area as well as counts of overflow cylinders filled, unused tracks still available in the overflow area, accesses made to the overflow area, levels of index used and tracks needed to accommodate the high level indexes. If provision of input is from existing file data as in re-creation or from system useable files, the interrogation capabilities are available to monitor this activity but no special analysis reporting is provided.

TDMS

Errors detected in the automatic validation functions of GENERATE are reported at the users option either in a batch report or on-line permitting interactive correction of data during the population activity. The interrogation capabilities are not available during file population. After population is completed a program named CHECKER is available to check internal tables for correct pointers and associations. Data bases prepared by GENERATE can be processed by a program named SIZES which provides statistics about the file storage required to accommodate the data in the file.

UL/1

Errors detected in the data by the validation procedures specified by the user are reported. After population is completed by the ESTABLISHMENT function the schema is printed showing the item number, name, description and maximum length, also the maximum number of multi-valued items and statistics on how the data is stored.

7. PROGRAMMING FACILITIES

Programming facilities are so named because they can only be accessed through a procedural program written in a conventional programming language, called the host language. The user can only invoke the programming facilities by writing and executing a program; therefore, he is called a programming user. The facilities provided by a system for the programming user represent capabilities which are distinctly different from the functions of Data Definition, Interrogation, and Update.

In writing a program, the user calls upon the facilities by issuing a statement. The statement may be an explicit CALL with associated parameters or it may be incorporated as a macro or a verb in the host language. In the statement the programming user specifies which facility he wants such as read, write, replace, or open, and any other parameters required by the system such as error control and buffer designation.

The statement types used to call upon the programming facilities do not constitute a complete language in themselves. Therefore, they must be embedded in a conventional procedural language such as COBOL, FORTRAN or assembly language, which is the host language.

In executing a statement, the system may perform some additional support activities such as security maintenance, backup, index maintenance, data definition maintenance, ordering maintenance, and run-time validation.

With the increased flexibility inherent in the use of programming facilities comes an increased responsibility for the programming user since there is a higher risk of destroying the integrity of the data base. The data manipulation language statements of the programming facility permit a more detailed and precise control over the flow of data to and from the data base stored on secondary storage devices.

Sometimes a self-contained system will permit the user to write and incorporate his own program to perform processing within narrow constraints. This is sometimes called the own code capability. It is like writing a subroutine which the system will execute at the appropriate place in the processing sequence. It constitutes an avenue whereby the user can interrupt the function processing to do some special processing which may be difficult or impossible to accomplish using the facilities of the self-contained function. Own code is usually used to manipulate the contents of a buffer after reading data from the data base or before writing data to the data base. The own code capability is not intended to be used to control data flow by calling upon the programming facilities; to do so results in an inherent opportunity to seriously disrupt the operation of the system. Therefore, any discussion of an own code capability is included with its associated function and not under programming facilities (see Chapters 4 and 5).

COBOL, DBTG, IDS, IMS

Of the two kinds of data base management facilities, programming and self-contained, these systems provide only programming facilities. The set of DML statements is called the Data Manipulation Language (DML) in DBTG, and Data Language/I or DL/I in IMS. There is no special name for the programming facility statements in COBOL or IDS, although in COBOL they constitute part of the set of input-output statements.

SC-1

The system provides both programming facilities and the functions normally associated with self-contained systems. The set of programming facility statements is called DAMOL, DAta Management Oriented Language.

7.1 Summary of data manipulation language statements

Figure 7-01 lists the types of statements associated with programming facilities and shows for each system the statements which correspond to each type. The terms used to designate DML statement types are used throughout this chapter.

STATEMENT TYPE	COBOL	DBTG	IDS	IMS	SC-1
CONTROL					
open	OPEN	OPEN	OPEN	-	OPEN
close	CLOSE	CLOSE	CLOSE	-	CLOSE
conditional	-	IF	IF	-	-
RETRIEVAL					
locate	SEEK	FIND	RETRIEVE	-	SEEK FIND
locate and access	READ	-	HEAD	GET UNIQUE GET NEXT GNP	READ OBTAIN ACCESS
simple access	-	GET	MOVE	-	-
hold	-	KEEP FREE	-	(HOLD option)	(THIS option)
currency reset	-	MOVE	-	-	RECORD SET
MODIFICATION					
add	WRITE	STORE	STORE	INSERT	WRITE
change	-	MODIFY	MODIFY	REPLACE	REPLACE
delete	-	DELETE	DELETE	DELETE	DELETE
reorder	SORT	ORDER	SORT	-	-
reorganize	-	INSERT REMOVE	-	-	-
SPECIAL					
table handling	SET SEARCH	-	-	-	-
communications	RECEIVE SEND ENABLE DISABLE ACCEPT DISPLAY	-	-	GET UNIQUE GET NEXT INSERT	-

Figure 7-01
Summary of data manipulation language statements

7.2 Modes of processing

In processing the data base the programming user may be required to explicitly declare the mode to be used. The two common mode distinctions are between input, output, and update, and between random and sequential processing. Depending on the mode, certain statements may be inoperable or some statements may require the user to initialize certain communication items in advance.

7.2.1 Input, output, and update modes

One common mode distinction is between input, output, and update. The programming user may declare his intention with respect to each file or area to be processed by his program. The declaration may be made in the open statement or prior to initiating execution of the program.

Under input mode the user indicates that he only intends to retrieve data from some named portion of the data base. Data will only be transferred from the data base to the program. Any statements which call for a modification of the data base would not be executed.

Under an output mode declaration the programming user declares his intention to only transfer data from the program to the data base. The output mode is used to perform initial file creation or, in some cases, to add entries to the end of an existing file. Adding entries to the end is quite distinct from inserting entries in the file so as to maintain any established ordering. Only the add type of modification statement is operative on a file in output mode in those systems providing an output mode.

Update mode encompasses both input and output. Updating a sequential (see 7.2.2) file requires that a new copy be created. Therefore, two files can be defined, one declared for input and the other for output. This approach is taken by COBOL and IMS so that the update mode is reserved for random files. Alternatively, the SC-1 approach is to permit an update declaration with either a random or sequential file. In the latter case, a new file for output is automatically declared by the system. The system keeps track of generations of a file.

The system's term for each mode is given in Figure 7-02.

SYSTEM	WHEN MODE DECLARATION IS MADE	INPUT	OUTPUT	UPDATE
COBOL	at open	INPUT	OUTPUT	I-0 (random only)
DBTG	at open	RETRIEVAL*	-	UPDATE
IDS	at open	RETRIEVAL	-	UPDATE*
IMS[1]	independent of program and prior to execution (when "PCB" is defined)	G (GET)	L (Load)	A (all)[2] or any combination of I, R, D. (random only)
SC-1	at open	INPUT	OUTPUT (sequential only)	UPDATE (copy created if sequential)

* The asterisk is used to designate the default option, which is selected by the system when the user makes no declaration.

[1] Processing mode declaration can be made for the file or for each group within the file.

[2] The update mode declaration can be restricted to any subset of the three modification statements.

Figure 7-02
Input, output, and update modes

7.2.2 Random and sequential modes

A second mode distinction is frequently made between sequential processing and random processing. Under sequential processing it is assumed that the system knows and maintains some sequential relationship among entries in a file or members of an assembly. The user then must, at least conceptually, reference entries in that sequence. Referencing the "next" entry is always meaningful under sequential processing, and some systems provide for referencing the "prior" entry.

Under random processing it is usually assumed that the user can reference any entry regardless of what was last referenced. In this case some criteria is needed by which to identify the desired entry or entries. It may be a single unique identifier or a complex conditional expression (see 7.4.4).

Even in sequential processing, some selection criteria may be used, but if so, a "forward" search through the sequence of entries from the current position is generally implied.

A file stored on a sequential medium (that is, a physically sequential file) can generally be processed only in a sequential mode. Processing a sequentially stored file in random mode would be quite inefficient and is not allowed in most systems. Sequential processing can generally be performed on either random or sequential access files. From the viewpoint of the user, a file declared for random processing can still be processed in some pseudo-sequential mode.

COBOL

The processing mode for each file is declared in the File Control paragraph of the Environment Division. The relevant clauses are:

$$\left[\ ,\ \text{ACCESS}\quad\text{MODE}\quad\text{IS}\quad \begin{Bmatrix}\text{SEQUENTIAL*}\\ \text{RANDOM}\end{Bmatrix}\right]$$

$$\left[\ ,\ \text{PROCESSING}\quad\text{MODE}\quad\text{IS}\quad \begin{Bmatrix}\text{SEQUENTIAL*}\\ \text{RANDOM}\quad\text{FOR}\quad\text{integer}\quad\text{RECORDS}\end{Bmatrix}\right]$$

$$\left[\ ,\ \text{ACTUAL}\quad\text{KEY}\quad\text{IS}\quad\text{item-name}\right]$$

For files stored on direct access devices ("mass storage") both the access mode and processing mode clauses are required. The "actual key" clause is required for random access. Also, random access is a prequisite for random processing. Thus, there are three modes of access-processing: sequential-sequential, random-sequential, and random-random.

Under sequential access, entries stored on a direct access storage device are obtained or placed sequentially. Under random access, the location of the entry to be obtained or placed is found using the item named as the "actual key".

For sequential processing, the entries are processed in the order in which they are accessed. Furthermore, they are processed one at a time.

A random processing declaration enables the system to process, asynchronously, more than one entry at a time. The number is determined by the integer in the random processing clause. The asynchronous, random processing of entries on direct access storage, requires that the user write the associated procedural statements with a USE statement in the Declarative section of his program. The out-of-line procedure so defined is invoked using the PROCESS statement.

Although the "actual key" item is optional for sequential access, if one is declared, the system will update it:

- following a WRITE statement execution; and
- prior to a READ statement execution only if not logically preceded by a WRITE statement execution.

The "actual key" data item is never referred to or required by the system if sequential access is declared.

DBTG

No mode distinction is made by the user between sequential processing and random processing. However, the proposed system is oriented to files on direct access storage devices.

IDS

Every file is stored on direct access storage devices and therefore random processing is always the mode.

IMS

The definition of a file is made independently of any processing program and stored in a "Date Base Description (DBD) Control Block." At that time, the access method to be used for the file is declared as either random or sequential. Sequential processing can be declared in the "PCB" (see 7.4) for input or output, otherwise random processing is assumed.

SC-1

When a file is opened the user specifies whether random processing will be used. This option is only possible on input or update. If update, the file must be stored on a direct access storage device. For input mode, the file may be stored either on a direct access device or on a sequential storage medium, although random processing, which allows stepping back in the file, of a sequential file is rather inefficient.

For sequential processing, from the time a file is opened until it is closed, only a strict "forward" motion through the data base is permitted. Therefore, the statements FIND, OBTAIN, RECORD, and SET are not executable.

7.3 Method of interface

7.3.1 Invocation of facilities from the host language

The way a programming user may invoke a programming facility varies. One way is to incorporate statements directly into the host language using a set of verbs which are recognized at the time of compilation. In this case, either a precompile is performed or the host language compiler recognizes and processes data manipulation language statements. Where the host language is an assembly level language, the programming user may use a set of macro instructions.

Another way to invoke the facilities is through the use of a CALL statement. The program may call a single controlling module which interprets the statement and then passes control to the appropriate facility module. Alternatively, the desired facility module could be called directly by the program. The calling sequence may include such data as the facility desired, the user working area, and the selection criteria.

In some systems, the normal sequence of host language statements may have to be explicitly interrupted by preceding the invocation of programming facilities with a control statement which indicates to the compiler or processor that it must "exit" the host language to interpret the statement or set of statements. The exit statement may serve to signal a precompiler that action is required on the statement which follows.

Some systems enable the programming user to call upon the facilities from programs written in different host languages. In some cases, the form of the language statement and the method of interface with the system will be the same. In other systems the method of interface may be different from one host language to another.

In a CALL statement, one of the parameters of the calling sequence identifies the desired programming facility by pointing to a data manipulation language statement. With the statement one step removed from the host language, it need not vary in syntactic form from one host language to another.

On the other hand, when a verb is used, the form of the DML statements will have to conform to the same general rules relating to the host language. The differences among host languages may or may not materially affect the method of interface.

SYSTEM	HOST LANGUAGES	METHOD OF INVOCATION	EXPLICIT EXIT REQUIRED FROM HOST LANGUAGE
COBOL	n.a.	verb	no
DBTG	COBOL	verb	no
IDS	COBOL (FORTRAN)	verb	yes, "ENTER IDS ... ."
IMS	COBOL PL/I BAL	call to control module	yes, only in COBOL: "ENTER LINKAGE ⋮ ENTER COBOL"
SC-1	COBOL PL/I FORTRAN BAL	call to control module	yes, only in COBOL: "ENTER LINKAGE ⋮ ENTER COBOL"

Figure 7-03
Invocation of facilities from the host language

COBOL

This is itself a host language. Nevertheless, it does provide control, retrieval, and modification statements. They serve as a basis of comparison with the data manipulation language of other systems.

DBTG

The current proposal presents a data manipulation language for COBOL. Nevertheless, the programming facilities could in principle be invoked from any host language.

IDS

The programming facilities are called by writing statements directly in a COBOL program. The set of data manipulation language statements must be preceded by an explicit exit from the host language in the form:

ENTER IDS statement-1 [; statement-2]... .

statements 1 and 2 cannot be regular COBOL statements and they must be followed by a period.

The programming facilities are available to programs written in FORTRAN or assembly language but only when invoked by a subprogram which is written in COBOL.

IMS

A program invokes the programming facilities by calling a named entry point of a control module. There is a single control module and it can be called from COBOL, PL/I, or assembly language. The appropriate facility module is incorporated into the program by the loader. If the single control module is overlaid, it will be reloaded when next referenced.

In COBOL an explicit exit is required using "ENTER LINKAGE". When returning, "ENTER COBOL" is required. There is no explicit exit from PL/I or assembly language.

Within each host language the sequence of calling parameters consists of the addresses of:

- type of facility
- file name
- user working area
- up to fifteen selection criteria (not always required)

In addition, PL/I requires a fifth parameter to precede these. It is a count of the number of parameters in the calling sequence.

In the program the user must declare each of the parameters which are to appear in the calling sequence. For example, in COBOL, in the Working Storage Section

```
      77  LABEL  PICTURE  XXXX  VALUE  'GUƀƀ'.
```

defines LABEL with the value for the GET UNIQUE verb. All data manipulation language verbs must be defined left justified in four character variables.

In COBOL, the user is also required to name all the files that are referenced in the program:

```
      ENTRY   'DLITCBL'   USING   file-name-1  [,file-name-2]...
```

This statement must contain the same number of file names in the same sequence as there are "PCB's" in the "PSB". The same information must be provided in the PROCEDURE statement of PL/I.

SC-1

To obtain the programming facilities a program must call the control module "DSKERNEL", a subprogram incorporated into the program by the loader. The control module cannot be overlaid and therefore must be in the root segment of a large segmented program.

In COBOL an explicit exit is required using "ENTER LINKAGE". When returning, "ENTER COBOL" is required.

The call statement is the same for FORTRAN, COBOL and PL/I programs. Due to slight differences in the handling of subroutine linkage by the PL/I compiler, a separate entry point to the control module is used by PL/I programs. The sequence of calling parameters is the same for all four host languages.

The calling sequence consists of the addresses of three parameters:

- the "DAMOL" statement
- the user working area (not always required)
- an error control word

The "DAMOL" statement is declared and stored as data in alphanumeric format in the user program, enabling it to be dynamically modified by the user program between executions. The statement must be scanned and interpreted each time it is encountered during program execution.

7.3.2 Language form

The set of statements specifically provided to invoke the programming facilities constitutes the data manipulation language (DML). The data manipulation language is usually narrative. This is possible whether the DML statement is incorporated directly into the host language or referenced by a parameter in the calling sequence. Where the statement is a host language verb, the form of the syntax will generally follow that of the host language.

COBOL, DBTG, IDS

Language form is narrative.

IMS

In the calling sequence the language form is separator. In the selection criteria the language form is fixed.

SC-1

In the calling sequence the language form is separator. The language form of the DML statement, which is addressed in the calling sequence, is narrative.

7.3.3 Addressable data structures

In some systems it may be impossible to address, or there may be certain restrictions on addressing, certain types of data structures in the DML statements.

In the syntactic presentation of the various DML statements in later sections, it may be convenient to use certain conventions for some systems. When two reserved words are permissible but equivalent, only one is shown.

COBOL

An open statement references a file. The access and modification statements refer to an entire entry. The MOVE statement is available only to manipulate data within the user working area and can operate on items or groups.

A data item name may be subscripted (see 7.5.4.1) or qualified or both depending upon its definition. In COBOL it is called an "identifier" in this general case. However, for simplicity, item-name will be used in the subsequent presentation of specific DML statements.

DBTG

A group relation ("set") or an "area" can be referenced by an open statement. An "area" can contain instances of one or more groups, which constitute part or all of a group relation or multiple group relations.

The locate statement FIND addresses a group ("record") and the simple access statement GET addresses the found group or items within the group. All the modification statements address a group, except for MODIFY which can also address items within the group.

A data item name may be subscripted or qualified or both. If the same item name is used in more than one group, qualification using group name may be necessary to achieve uniqueness. Subscripts are used when reference is made to a data item within a data aggregate defined with an OCCURS clause. The subscript is an integer greater than or equal to 1. For simplicity, item-name will be used to represent all of these possible cases.

IDS

An open statement references a file. Statements always refer to groups or group relations, except for the change statement and the MOVE statement which can refer to principal items (declared at the 02 level).

IMS

Statements can refer only to groups. In defining a file one identifier ("key") item can be declared for each group. The programming user may define multiple "logical" files which permit the use of an alternate identifier for groups not at the root level of the structure. The identifier item also functions as a sequencer item.

SC-1

In the current release the open statement references a file. In a future release a conditional expression (see 7.4.4) will be permitted with the open statement. In this way the open will be able to reference an assembly. The conditional expression identifies instances of parent repeating groups and the parent entry for the assembly in the data structure. A conditional expression may also select entries or groups in the referenced file or repeating group schema.

The access and change statements can reference a group or an item. The locate and add statements reference a repeating group. The delete statement can reference any data base sturcture or element.

All data structure names refer to the subschema. The "Bindlist" provides the correspondence with data base names.

In the presentation of each DML ("DAMOL") statement the following abbreviations are used:

group/item-name	name of a repeating group, a non-repeating group, or an item. If a file is only one level, then group and entry would be synonymous.
rgroup-name	name of a repeating group.

7.4 Program-system communication

When issuing a DML statement certain information must be communicated to the system. This includes the facility desired, the data structures to be referenced, the criteria to be used in selecting data structures, the location of the user working area, and an indication of what to do upon detection of an error or exception condition. This information may be provided explicitly as part of the statement or calling sequence or it may be implicitly assumed by the system. The information assumed by the system may be based upon conventions which are understood and to be followed by the programming user or they may be based upon a context which was defined or established earlier through prior statements, declarative statements in the program, the data description section of the program, or the data definition function.

In the case of a DML statement, explicit information is provided in the form of operands and clauses. The call statement would have associated with it a calling sequence of parameters. In addition to the explicit information, a number of data item names may be established to communicate certain information between the program and the system. One use for such additional communication parameters is for the system to keep track of the current position in the data base. The pointer mechanism and methods of manipulation aid in understanding the operation of the programming facilities.

What the system remembers or assumes between the execution of DML statements should be known by the programming user since it may dictate to some degree the sequence in which the statements can be issued.

The word 'system' in program-system communication refers to the system module, sometimes called the run-time support module, which first receives and interprets the DML statement for execution. The communication parameters may be used by any module called under the control of the run-time support module. Program-system communication does not refer to the transfer of parameters between two user-written host language porgrams.

After reviewing the overall approach to program-system communication, specific subjects are covered in subsequent sections. These are the concept of currency, the method of defining and handling the user working area, handling error and exception conditions, the scope and method of presenting selection criteria, and programming user requirements for security clearance. These topics cover elements which are common to more than one DML statement. Specific DML statements are referred to even though their precise definition appears later (see 7.5).

COBOL

All communication takes place in the Environment Division, the Data Division or directly in the DML statement. The DML statement in general contains:

- verb for the facility requested
- file name
- other parameters relating to the verb.

DBTG, IDS

In addition to the parameters of the DML statement, special communication items are established to permit additional program-system communication. These "system communication locations" are referenced as variables in the host language program.

IMS

In addition to the information contained in the calling sequence of the DML statements, information is communicated in the "Program Specification Block (PSB)". The "PSB" contains at least one "Program Communication Block (PCB)" for each file to be referenced by the program.

The "PCB" contains the following information:

1. file name (the "PCB" identifier; the same name that is used in data definition).
2. group schemas to which this program is bound ("sensitive").
3. processing mode for each group schema indicating:

   -input, output, or update (see 7.2.1)
   -sequential processing only (on input or output)
   -processing done to the exclusion of all other users.

4. status code (error control word).
5. name of the group schema last processed by the system (current).
6. level in the hierarchy of the group last processed by the system.
7. maximum combined length of concatenated group identifiers ("keys").
8. concatenated group identifiers for all groups from the "root" group down the hierarchy to the group last processed by the system.

The "PSB" is generated independently of and prior to program execution. It describes the program and its use of the data base and terminals. For each "PCB" values are established for communication items 1,2,3 and 7 above. During execution the system has exclusive authority to change the other four items. The programming user is only allowed to access the communication items in each "PCB".

Within a program each "PCB" must be described. The descriptions must correspond exactly to those given at the time the "PSB" is generated.

An 01 level Linkage Section entry in the Data Division of a COBOL program defines an external data item for each "PCB". The EXTERNAL data attribute in PL/I performs the same function. Each of the communication items listed above (and some others of no consequence here) are declared at level 02. It is through this linkage that a program accesses the status code and other items after execution of a DML statement.

SC-1

All program-system communication is accomplished directly through parameters in the calling sequence and in the DML statement.

The communication elements are:

1. the DML statement (address is the first parameter of the calling sequence) which is in narrative form and contains:

    -verb for the facility being requested
    -referenced data structure
    -other parameters related to the verb
    -conditional expression (not always required)

2. user working area (address is the second parameter in the calling sequence).

3. error control word (address is the third parameter in the calling sequence).

7.4.1 Currency

When a program references the data base and the system successfully executes the DML statement, there exists, in effect, a pointer to the element which was referenced in the data structure. The element is usually an instance of a repeating group. The element now "pointed to" can be considered current.

Some concept of currency must be known to the programming user if he can issue any DML statement which depends upon position in the data base during previous processing.

Some systems actually store a pointer in one form or another and make it available to the program. The program may then save the currency pointer and use it to "reset" the system at some future time. This is one type of selection criteria (see 7.4.4).

When a structure is established as current and some sequence of similar structures is known to both the program and the system, the program may be able to reference the "next" or "prior" member of the sequence.

There may be multiple currency pointers at any given time. For example, the system may retain information concerning the current entry of a file, the current group instance in an assembly, the current group corresponding to a given schema (type), or the current group in a group relation. In this case, if the user wants to reference the current element, he would have to name the file, the group schema, or the group relation schema.

COBOL

Under sequential access a READ statement references the next entry in sequence. If an entry identifier item ("actual key") has been declared, the system updates it after a WRITE statement execution, and prior to a READ statement execution only if not logically preceded by a WRITE statement execution. For sequential access the entry identifier item is not used by the system, but it is updated and made available to the programming user.

Under random access, the desired entry is specified by the contents of an entry identifier item. This item is used by the SEEK statement (or, in its absence, the READ and WRITE statements). The item contents are preserved until the beginning of the next DML statement execution. The programming user can retain values of entry identifiers in other program variables if he desires to return to an entry that was current during previous processing.

DBTG

For the duration of each run unit (during execution of a program) the system maintains currency information on the identity of the most recent group processed (unless suppressed) by the run unit. Currency information pertains to:

1. "Current group of group name"--the current group within each group schema (type) known to the run unit.

2. "Current group of group relation name"--the current group within each group relation schema ("set type") known to the run unit. Note that the current of group relation may be any group type which participates in the group relation whether as a parent or as a dependent group.

3. "Current group of area name"--the current group within each area known to the run unit.

4. "Current group of run unit"--the current group known to the run unit (within any group type, group relation or area).

The system also maintains:

5. The name of the group schema to which the "current group of run unit" belongs; referred to as "RECORD-NAME" in the program.

6. The name of the area which contains the "current group of run unit"; referred to as "AREA-NAME" in the program.

The locate statement provides a variety of options for referencing groups based upon these currency indicators.

Each group in the system is assigned a unique internal identifier, called a "database-key". The currency pointers are maintained in terms of this internal identifier. When a group is added to the data base it is assigned a unique internal identifier which it retains until deleted from the data base or until the data base is reorganized.

IDS

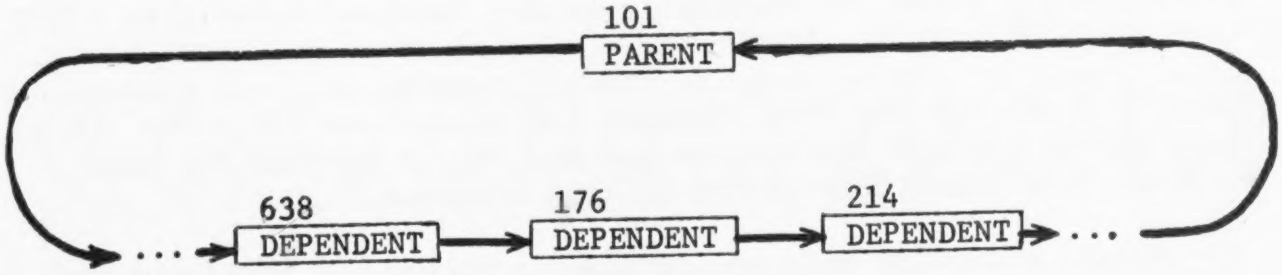
Each group ("record") is assigned a unique internal identifier ("reference code") which is permanent until the group is deleted. Currency records are maintained in terms of these internal identifiers.

The internal identifier of the current group (the one last processed) is always available to the program in the communication item named "DIRECT-REFERENCE". Thus the system provides the mechanism for the currency pointer to be reestablished at some previous location.

As the system moves through the data base it maintains a "chain table" for each group relation ("chain") in which a given group schema is defined. The programming user can reference these tables using the MOVE statement (7.5.2.5). Knowing what a "chain table" is and how it is used can aid in understanding the DML statements.

A "chain table" contains unique internal identifiers for a parent group, and for the current, prior, and next dependent groups of that parent in the group relation. As the system traverses a "chain", the table is updated with the internal identifiers of the groups. One "chain table" is maintained for each "chain" defined. Thus the system retains multiple currency pointers to the data base.

For example, if a group relation appeared as follows:



The table might appear as:

Parent	101
Prior	638
Current	176
Next	214

The system also maintains the internal identifier for the current (last processed) group of each group schema. The programming user can reset the currency pointers by issuing a RETRIEVE CURRENT RECORD statement (see 7.5.2.1).

IMS

The system always remembers the group last processed so that the user can call for retrieval of the next group. If a group is not named, the system delivers the group which is next in the sequence without regard for group schema (type) or level. The user then references the "PCB" (communication item 5) to determine which type of group was retrieved.

The system also remembers a current parent group. Successful execution of a GET NEXT or GET UNIQUE statement establishes the current parent as the parent of the group retrieved. By using the GET NEXT WITHIN PARENT statement, the user ensures that the system will not cross a hierarchical boundary to retrieve a group. That is, the system will only consider groups which are dependent (at any lower level) on the current parent group. Modification statements do not cause a change in the group established as current parent.

SC-1

Each structure be it file, entry, assembly, repeating group instance, non-repeating group, or item has a unique internal identifier ("IPC--item position code"). If the opened structure is declared random, this identifier can be saved by the programming user and subsequently used to reset the system's currency indicator (see 7.5.2.5). It is transferred between the program and the system through the user working area identified in the calling sequence.

The system maintains only one currency pointer. It distinguishes between pointing at a group or between two groups. After execution of a locate or add statement the system points between two groups. After execution of an access or change statement, the system points at the group just processed. A reprocess statement requires that the system point at a group prior to execution.

7.4.2 User working area

The locations in primary storage where data from the data base is made available to the user program during execution, or where the user program stores data which is to be transferred to the data base, is called the user working area. Conceptually, the user working area is a loading and unloading zone where all data provided by the system in response to a data retrieval statement is delivered and where all data to be picked up by the system must be placed. There is no implication that the user working area is contiguous. It may include named, non-contiguous locations.

Each concurrently running program may have its own user working area. Sometimes a user working area is associated with each OPEN statement executed.

The user working area is not to be confused with the system buffer(s) which are used to receive blocks of data from secondary storage. Both types of areas may be set up in the system but the data is directly available to the user program only in the user working area.

Creation of the user working area may be one of the products of the binding process. The data definition function describes the data as it is in the data base and how it appears in the system buffers. The auxiliary data definition in effect defines the user working area and how and where the data will appear.

Knowing the way in which the system handles the user working area and how the user must handle it is often necessary to an understanding of the operation of the programming facilities. Systems may vary in the method of handling the working area and in the extent of programming user involvement. Any conversion or transposition of data which occurs between the user working area and the system buffer during statement execution is considered part of the handling process. Also, some systems provide the ability to overlap or rotate parts of the user working area.

Another aspect concerns the way in which the user, programming in the host language, references the data items in the user working area. Some systems require an explicit move to take a value from the user working area and assign it to a variable which is known to the program before it can be used as an operand in, say, an arithmetic expression. Some systems may permit one or more definitions of data items in the working area to be used directly by the program in assigning values to a variable. In this respect it is important to know whether or not different definitions of data items can be related to the same physical storage in the working area. Other systems may view the working area as merely a string of characters and require the user to refer to the area with a pair of integers specifying which character relative to the beginning and how many characters to access.

COBOL

Each file to be opened in a program has associated with it one user working area to store an entry from the file, and one or more system buffers (areas). In the Working Storage Section of the Data Division the user defines each entry to be stored in the user working area. The File Description section specifies the groups and their structure that go to make up an entry of the file.

Buffers can be shared using the SAME clause in the I-O CONTROL paragraph of the Environment Division. A set of files can be specified to share the same entry storage area in the user working area or further, to share the same system buffers. Files which share the same system buffers cannot be in the OPEN state concurrently.

The INTO and FROM options can be used with the READ and WRITE statements respectively to designate a second area of storage to be used in conjunction with the user working area. With READ, the data will be put into the designated area and the user working area. With WRITE, the data will be taken from the designated area, placed in the user working area and then written to the data base.

The statement

$$\text{MOVE}\left\{\begin{array}{l}\text{[CORRESPONDING]}\quad\text{item-name-1}\\ \text{literal}\end{array}\right\}\text{TO}\quad\text{item-name-2}\quad\text{[, item-name-3]...}$$

transfers data to one or more data items in accordance with an extensive set of editing rules. When the CORRESPONDING option is used all identifiers must be group items. Using the INTO or FROM option in a READ or WRITE statement an implicit MOVE is specified to move a set of data items (not groups) from one area to another. Thus a dual data transfer takes place.

Under random processing a special "saved area" must be defined to contain the entries of each file. The following appears in the Data Division:

$$\text{SA}\quad\text{area-name;}\quad\text{AREA CONTAINS}\quad\text{integer-1}\left\{\begin{array}{l}\text{CHARACTERS}\\ \text{RECORDS}\end{array}\right\}$$

$$\left[\text{; RECORD CONTAINS [integer-2 TO] integer-3 CHARACTERS}\right].$$

and simply sets aside an area in primary storage. The user may specify the size in terms of a number of characters, in which case the saved area is able to hold a variable number of entries. Alternatively, a fixed number of entries can be declared in which case the fixed entry size (integer-3) or the minimum and maximum sizes must be specified for the entries.

Processing an entry of a random file is initiated by a PROCESS statement which initiates the execution of an out-of-line subroutine. The subroutine is specified at the beginning of the Procedural Division with a USE statement. One "saved area" is automatically associated at object time with each out-of-line processing cycle.

Only one processing cycle has access to a single entry in the "saved area" at any one time. Upon completion of the asynchronous processing cycle, the area is released for further storage assignment. The execution of a PROCESS statement makes the associated "saved area" entry unavailable for reference from any in-line procedure. The "saved area" entry is followed by a "record description".

The entry to be processed is identified in the statement:

$$\text{PROCESS}\quad\text{section-name}\quad\text{[FROM identifier]}\quad\left[\text{USING}\left\{\begin{array}{l}\text{area-name}\\ \text{group-name}\end{array}\right\}\right]$$

The FROM implies that a MOVE statement is to be executed; area name indicates that the user has already placed the entry in the named "saved area"; and "record-name" is the name of an 01 level entry which is subordinate to a "saved area" declaration.

The statement

```
     ┌ALL                                        ┐
HOLD {                                           }
     └section-name-1  [ , section-name-2 ]...    ┘
```

provides a delay point that causes asynchronous processing of all or named sections to be completed before synchronous processing is resumed following the HOLD.

DBTG

Each program has its own user working area. The data in the working area of a program is not disturbed except in response to a DML statement execution or by the user program's host language procedures. There is no implication that user working area locations are contiguous.

Data item values are moved from the system buffers to the addresses or variables defined in the host language (the user working area), only by issuing a "GET" statement.

The user working area is set up by the system in accordance with the auxiliary data definition ("subschema") invoked in the program. Each data item named in the "subschema" invoked will be assigned a location in the invoking program's user working area and may be referenced by its declared name. Where a "subschema" is invoked, the type of data items included in the "subschema" and, therefore, in the user working area, may differ from the type of those items in the definition for the data base. The system is responsible for required conversions. Data items included in the data base, but not in the "subschema" invoked, are not assigned space and cannot be referenced.

User defined procedures can be invoked whenever a data item requiring special conversion is retrieved or modified. The clause

```
    ┌ENCODING┐
FOR {        } [ALWAYS]   CALL   procedure-name
    └DECODING┘
```

is added to the item definition during data definition. Encoding is performed when an item value is transferred to the data base with a modification statement. Decoding is performed when an item value is transferred to the user working area with a retrieval statement. Both an encoding and a decoding clause may be used for the same data item. The procedure is invoked in lieu of a standard conversion procedure. If the ALWAYS is not used, the procedure is invoked only if the characteristics of the item vary between the schema data definition and the auxiliary data definition.

IDS

The user working area is defined using the facilities of the COBOL host language, in the Working Storage section of the Data Division. An area is created for each 02 level item or group. Data items subordinate to the 02 level are not addressable by DML statements. Dependent items and groups at the lower levels must be referenced by host language statements. The communication interface between DML statements and host language statements is the user working area established for each 02 level item defined in the Data Division.

When a group is retrieved from secondary storage, its data items are available to the user program only after they are explicitly moved to the user working area. Data item values are moved from the system buffers to the locations of variables named in the host language Data Division, only by executing a MOVE statement or a HEAD statement.

With a MOVE the user can move an entire entry, groups, or selected 02 level items. Each data item is unpacked from the system buffer and put into the user working area in accordance with the item schemas provided in the Data Division. When a current dependent group exists in a group relation, the HEAD statement is used to transmit all items of its parent group into the user working area.

Before storing a group, the user working area must first be initialized with the data item values of the group to be stored.

If the same 02 level item name is used in different group schemas they will share the same location in the user working area.

IMS

The user working area ("segment input-output area") is declared in the host language program. The 01 level specifies the name of the area which is subsequently used as the third parameter in the DML statement calling sequence. Items declared at the 02 level and lower can be referenced by other host language statements. The system does not maintain a definition of item schema within a group except for identifier items. Thus two programs which reference the same group schema must both use exactly the same user working area definition. The system performs no conversion.

The user may desire an area to be shared by more than one group schema (type). If a common area is used, it must be as long as the longest group to be processed. The group is always left-justified within a common working area. The name refers to the leftmost position of the area.

SC-1

Specification of a user working area is required by the access, add, and change statements. An address in primary storage is given as the second parameter in the calling sequence. This working area is also used by the currency reset statements to contain a unique internal identifier.

Within his program the user declares space for one or more working areas in the manner normally provided by the host language.

The user must define his working area to the system before he can OPEN and manipulate the file. This is done using the "Data Bindlist Definition" function (see Chapter 3) which processes the auxiliary data definition. The auxiliary data definition always refers to a tree structure and the working area is a linearization of the data items in the structure (see Chapter 9). To achieve word alignment, the user can insert filler items of arbitrary length.

There is one user working area defined for each opened file.

The INTO and FROM options are provided with the access, add, and change statements, respectively, to reduce the amount of space a user need attach to his program for a given file. These options cause the system to modify the relative addressing scheme. Thus the user achieves an overlapping use of his working area. It must be defined at least as large as the largest group the user intends to process for the particular auxiliary data definition.

During execution of the program, if an item schema in data definition and auxiliary data definition differ in type or size, the system performs any required conversion of item values being transferred between the data base and the user working area.

The auxiliary data definition can differ from the data base structure in two additional ways. Items within a group may be transposed or omitted. Secondly, a hierarchy of groups can be flattened so that the programming user appears only to deal with a single level file, that is, a file in which all items are principal items.

7.4.3 Error and exception conditions

In any system error and exception conditions arise which may be communicated to the programming user and which may result in abnormal action by the system. A system often tests for a large number of conditions. Some conditions are obviously implied in the discussion of a particular DML statement and need not be called out. Greater concern rests with the method of handling error and exception conditions.

The approach of the system to handling errors can involve the programming user to a varying degree. One method of handling error or exception conditions is to return to the run unit with a value having been placed in a special communication item. Alternatively, the system may automatically branch to a specified error address where a code value can be examined and a prescribed out-of-line procedure executed. Some error conditions may result in the termination of the calling run unit issuing the DML statement.

Exception conditions include end of file and invalid group identifier. The system may allow the programming user to specify certain conditions which are to be detected and to specify the action to be taken upon the detection of certain conditions.

COBOL

The system provides a standard response to errors. No explicit provision is made within the language to allow user control over specific error conditions. The user may specify his own procedure to be executed after the standard input-output error routine. The statement:

$$\text{USE AFTER STANDARD ERROR PROCEDURE ON } \left\{ \begin{array}{l} \text{file-1 [, file-2]...} \\ \text{INPUT} \\ \text{OUTPUT} \\ \text{I-O} \end{array} \right\}.$$

at the beginning of the Procedural Division is followed by an additional error handling routine.

When the exception conditions of end of file or invalid entry identifier ("key") are detected, the system executes an imperative statement provided by the user as part of the DML statement.

DBTG

Six communication items are used for error conditions:

ERROR-STATUS: code indicating the type of error first detected and the DML statement in which it occurred.

ERROR-SET: group relation name, if appropriate, in which the error occurred.

ERROR-RECORD: group name, if appropriate, in which the error occurred.

ERROR-AREA: the "area" name in which the error occurred.

ERROR-TYPE: indicates the type of alteration performed by a concurrent run unit, if any.

ERROR-COUNT: contains the total number of errors detected during the system's attempt to execute the DML statement. When more than one error is detected the first four communication items above reflect the first error condition detected.

The values in the six error communication items remain unchanged until the execution of the next DML statement.

Error status codes are provided to indicate the exception conditions:

- end of group relation,
- end of "area," and
- no group was found which satisfied a given identifier or selection criteria.

The programming user has two possible avenues for specifying actions to be taken when the system detects an error. These actions are specified in addition to the standard error procedures provided by the system and they are executed prior to the standard procedures.

The USE statement specifies the error condition codes under which a user supplied procedure is to be executed:

```
USE ON ERROR [IF ERROR-STATUS EQUAL integer-1 [, integer-2]...]
```

The user supplied procedure is written immediately following the USE statement. Each integer value indicates a DML statement and error code combination for which the procedure is to be executed. If no error status codes are listed, the procedure will be executed for all error conditions that arise. Multiple USE statements are permitted but the same error status code may appear at most once.

Alternatively during data definition the programming user can specify that certain actions are to be taken whenever certain operations are performed on the data base. The execution of a named, user-defined procedure can be called for:

1. when an "area" is opened or closed,
2. when a group is the object of all or some retrieval or modification statements,
3. when an item is involved in a GET, STORE, or MODIFY statement, or
4. when a group relation is the object of a reorder or a reorganize statement.

IDS

The statement:

```
     IF ERROR statement-1 [ELSE statement-2  [; statement-3]...]
```

tests the occurrence of any logical error resulting from the last statement executed. It may only follow the statements: STORE, RETRIEVE, MODIFY, DELETE, HEAD and MOVE. Statement-1 may only be an unconditional transfer or a subroutine call. The specific errors which may occur are a function of the statement executed. The user program may determine the type of error by referring to the communication item named ERROR-REFERENCE.

Testing for data dependent error conditions must be incorporated in the procedural logic of the user program.

If the error occurs because of a hardware failure, data description error, or an improper use of DML statements, the program will be brought to an orderly halt, the file closed and the program terminated and memory dumped, if requested, with an appropriate error message.

The execution of a subsequent DML statement will reset the value stored in ERROR-REFERENCE.

The statement

```
     USE  procedure-name-1  [ THRU  procedure-name-2 ]

        [ WITH  TRACE ]

           { error-code-1  [ , error-code-2 ]... }
        ON {                                      }
           { ANY ABORT                            }
```

is used to specify procedures to be executed for specified or all error conditions which are in addition to standard error procedures. The USE statement can appear anywhere in the program. The procedures may not contain DML statements. The error codes used are system defined.

The trace option will cause the system to print out the names of the called subroutine and the calling routine repeatedly up the calling hierarchy until the main program is reached. All fatal error conditions result in a trace prior to termination of the program.

IMS

The "status code" in the "PCB" is used to communicate error or exception conditions to the program. It is available for interrogation upon completion of a call. The code remains there until another DML statement references the same file.

A blank status code indicates successful execution.

Warning codes may be given to a program following the execution of a GET NEXT type of statement (GN, GNP, GHN, GHNP) without a selection criteria. This can occur under two conditions. The first occurs when the system passes from one group in the data base to another which is at a higher level in the hierarchy (closer to the root). The second condition occurs when the system passes from one group schema (type) to another on the same level of the hierarchy.

Error codes are returned under a wide variety of conditions: for invalid or inconsistent DML statement calls, a group is not found for a given group identifier, end of file, an attempt is made to change the sequencing identifier in an existing group, the addition of a group would cause duplication of sequencing identifiers, and insertion of a group at the current position would destroy the ordering.

Under some conditions a program is not permitted to begin execution. For example, the auxiliary data definition in the program ("PSB") conflicts with the data base definition ("DBD").

SC-1

Certain error or exception conditions encountered by the system are passed back to the calling program in the location designated by the programming user. These conditions, if ignored or misinterpreted by the programmer, will not affect the integrity of the system (although they may cause trouble to the program). For example, the conversion of data items on transfer between the user working area and the system buffer can give rise to an error condition.

Other conditions are such that the programming user cannot always be depended upon to have provided adequate recovery action. This type of error results in termination of the calling run unit. The system prints an error message identifying the module and the error before terminating the program.

Many error messages, exception condition messages, and comment messages are provided by the system. They are classified as:

0 Success
1 End-of-file
2 No group for the given internal identifier (in currency reset)
3 Truncation
4 Code conversion } on transferring items between system buffer and user working area.
9 No operation; statement not executed.

The system logs internally all messages generated during execution of a run unit and prints them all upon completion or termination of the run unit.

7.4.4 Selection criteria

The selection of data in the data base may be accomplished by associating an identifier or a conditional expression with a DML statement. The conditional expression capability for selection criteria may be the same as that used in the interrogation function (see Chapter 4).

Selection criteria are only used to select from a set of data elements which replicate, namely, to select one or more entry instances from a file or one or more group instances from a repeating group or a group relation. Furthermore, selection criteria do not include selection based solely on the position of the entry or groups with respect to the current or last one processed, or to its position with respect to the whole set of replicated structures. Thus criteria such as NEXT, PRIOR, FIRST, LAST are excluded. If desired, they are included as options in the DML statements themselves.

The form and content of the selection criteria may be quite simple and restricted or it may be very flexible permitting such things as comparison of item values, to constants, ranges, other item values, or conditions of existence or type, and the combining of atomic conditions using the full power of logical and relational operators. In the simplest case, the selection criteria consists of a value for a unique internal identifier of a group.

Under random processing some selection criteria are required since selection of a group is not always based on a sequence relationship. Under sequential processing, selection criteria can still be meaningful. Rather than always access the next instance, the user may want the system to search through the set of replications beginning from the first or the current instance. The system would search through the sequence until an instance was found that satisfied the selection criteria or until the end was reached. In fact, if the file is on direct access storage the system may be able to go directly to the desired instance without a sequential search.

COBOL, IDS, DBTG, and IMS are similar in that they achieve selection through communication items which are initialized by the programming user with values of identifier items. In SC-1, a conditional expression is contained in the DML statement.

COBOL

Under random processing the user provides the entry identifier in the variable declared as ACTUAL KEY. The entry identifier is used by the SEEK statement (and the READ and WRITE in the absence of a preceding SEEK).

DBTG

The DML does not provide for selection criteria in the form of Boolean expressions. However, it does provide selection capability on data items specified in data definition with values supplied prior to issuing DML statements. In this sense a selection capability is provided where the criteria are always equal comparisons and united conjunctively.

The definition of a group specifies that its selection is "direct", "calculated" or via the parent of a group relation. If direct, a communication item must be defined to contain the unique internal identifier prior to execution of a DML statement. If calculated, one or more communication items must be defined to contain identifier values on which the location of the group is calculated.

If the selection of a group is via its parent in a group relation, the selection takes place in two steps -- first a unique parent must be selected, that is, an instance of the group relation, then the desired dependent group is selected.

The selection of the parent group is declared when the group relation is defined. It is accomplished in one of the three ways specified previously. In the group relation definition special item names must be declared which correspond to the communication items in the previous paragraph.

The dependent group can be found by declaring "SEARCH KEYs" which are identifier items from the dependent group. The system then searches for the desired group based upon the values of these identifier items which are supplied by the program prior to DML statement execution.

Selection of groups based upon values of these various identifier items is performed by the DML statements FIND, MODIFY, and STORE.

IDS

Selection criteria do not explicitly appear in the DML statement. The method of retrieval is largely determined at data definition time.

When an entry is defined, a special communication item is established to contain a unique internal identifier. This item is initialized prior to retrieving an entry. The programming user supplies a unique internal identifier in the communication item, "Direct-Reference", and then issues a RETRIEVE DIRECT statement.

A "calculated" group is retrieved by calculating an internal "page" address using the values of one or more identifier items in the group. These values must be provided by the user prior to retrieving a "calculated" group.

A dependent group may be obtained by initializing one or more identifier items (ASCENDING/DESCENDING KEY items) for the dependent group, and one or more identifier items (MATCH-KEY items) before issuing a RETRIEVE group-name RECORD statement. In the definition of the dependent group, the SELECT UNIQUE MASTER and the ASCENDING/DESCENDING options would both have to be specified. If the ASCENDING/DESCENDING option is not used, the first dependent group will be obtained.

When a dependent group is current, the parent group may be obtained by issuing a RETRIEVE MASTER or a HEAD statement.

IMS

The addresses of the selection criteria are the fourth to the eighteenth parameters in the calling sequence. Each selection criteria refers to one group at each level in the hierarchy down to the desired group. It contains a single condition on a single item in each group in the hierarchy.

Selection criteria are required on a GET UNIQUE statement and an INSERT statement, optional on a GET NEXT statement, and not allowed on DELETE or REPLACE statements.

Selection criteria consist of a sequence of conditions where the sequence corresponds to the hierarchy of groups, starting from the root group and going down to the desired group.

A condition has the following general form:

$$\text{group-name} \left[(\ \text{item-name} \quad \text{relation} \quad \text{literal}\ ) \right]$$

The group and item names must correspond exactly to those given in the data definition function. They must be eight characters long and right filled with blanks if fewer characters are used. The relational operator can be one of (where ƀ is a blank):

ƀ= ƀ> ƀ< ¬= => =<

The literal is the value against which the value of the named item will be compared based upon the relational operator. The literal value must conform to the type of the named item and be the same length. All comparisons are made on a logical bit-by-bit basis.

In some situations only the group name is specified. This is equivalent to other systems in which the DML statement contains the file name and the name of a group within the file.

The condition ("segment search argument - SSA") must be declared in the host language program. It can be declared as a unit or in pieces thus enabling the program to reference and change parts of the condition. The following is an example from COBOL.

```
01  COLLEGE-SSA
    02  SEGMENT-NAME   PICTURE A(8)     VALUE  'COLLEGEƀ'.
    02  LEFT-PAREN     PICTURE X(1)     VALUE  '('.
    02  SEGMENT-KEY    PICTURE A(8)     VALUE  'COLLEGEƀ'.
    02  RELATION       PICTURE X(2)     VALUE  'ƀ='.
    02  KEY-VALUE      PICTURE A(12)    VALUE  'MICHIGANƀƀƀƀ'.
    02  RIGHT-PAREN    PICTURE X(1)     VALUE  ')'.
```

Existence conditions, as such, are not supported by the system, but naming a group without a condition will result in an error code if no group satisfies the search; this can serve as an existence test.

Compound conditions at a single level are not possible. It is necessary to select a group on one condition and test other conditions on the selected group using statements of the host language. Only sets of conditions from a single hierarchical path are allowed and these may only be connected conjunctively.

Variations on the construction and use of selection criteria are discussed with each DML statement.

SC-1

A conditional expression is provided as the last clause of a DML statement. It is required with FIND, SEEK, and OBTAIN, and optional with OPEN, READ, REPLACE and DELETE. The condition is built up from atomic conditions of the form:

$$\text{atom} := \text{item-name} \;\; [\text{ IS }] \left\{ \begin{array}{l} \left\{\begin{array}{l}\text{EQ}\\ \text{GE}\\ \text{LT}\\ \text{NE}\\ \text{GT}\\ \text{LE}\end{array}\right\} \left\{\begin{array}{l}\text{literal}\\ \text{item-name}\end{array}\right\} \\ \qquad [\text{NOT}] \left\{\begin{array}{l}\text{NUMERIC}\\ \text{ALPHABETIC}\\ \text{PRESENT}\\ \text{POSITIVE}\\ \text{NEGATIVE}\\ \text{ZERO}\end{array}\right\} \end{array} \right\}$$

The conditional expression then takes the form:

$$\text{cond} := \text{WITH} \;\; [(]\ldots \;\text{atom}\; [)]\ldots \left[\left\{\begin{array}{l}\text{AND}\\ \text{OR}\end{array}\right\} \;[(]\ldots\text{atom}\;[)]\ldots \right] \ldots$$

where conventional rules for the use of parentheses apply. The AND and OR operators have the same level of precedence. To interrupt the normal left to right scan and execution, parentheses must be used.

In the current release a conditional expression can only refer to a single level, that is, one repeating group at a time. A future release will permit a conditional expression to refer to multiple consecutive levels in a single hierarchical path in the data structure.

7.4.5 Security clearance and integrity

Security clearance refers to the actions which a system takes in response to or in conjunction with the execution of DML statements. It also refers to the actions or information that the system requires of the user in issuing DML statements.

Actions relating to security clearance can take place at various levels. The broadest level is related to the auxiliary data definition. If a binding process takes place in the system to link the program to only part of the data base, thus making it impossible for the program to reference any part of the data base which is not known to it through the auxiliary data definition, then to a large extent this automatically ensures the privacy of the rest of the data base from that program.

Of more interest to the programming user is the security checking procedure that he will have to satisfy in order to have his DML statement executed. Security clearance may take place at the time of opening a part of the data base for processing and/or at the time of issuing DML statements. The security restriction may be applied to data access or data modification or to both. It may be based on some authority level and/or need-to-know. Security restrictions may be defined at various levels in the data base -- file, entry, group, item. Security also involves resolving the conflict when two concurrent programs desire to reference the same data, with at least one desiring to make modification.

The security clearance procedures will be closely associated with the security restrictions specified in data definition (see Chapter 3). Security from a system wide viewpoint is a Data Administrator function (see Chapter 8).

COBOL

none

DBTG

Provision is made for protection of privacy against unauthorized access of data, and for safeguarding the integrity of data from untoward interaction of programs.

PRIVACY LOCKS can be defined at any of six levels from the data base down to the data item. The programming user must provide PRIVACY KEYS in seeking to access or modify data which is protected by means of privacy locks. Locks can be specified against the use of specific DML statements.

When required by data definition, the program must provide keys to do any of the following:

- regular or exclusive retrieval or update of named "areas"
- INSERT, REMOVE, STORE, DELETE, GET, MODIFY, or FIND named groups
- GET, MODIFY, or STORE named data items
- ORDER, FIND, REMOVE, or INSERT named group relations

The key can be given directly as a literal, as a named data item to appear in the user working area, or as a named procedure which generates the key. The specification of these privacy keys is part of the identification division of the host language program. A privacy lock procedure can also be used to do such things as validate a privacy key, ask questions of a person at a terminal, terminate the run unit for repeated privacy violations, or disconnect the user terminal and call back before granting access.

To safeguard the integrity of the data, the system will not execute a modification statement following a FIND if a concurrently running program has executed a successful modification in the interim on the same group instance. At this point the current run unit can choose to execute the modification statement regardless, or to re-access the modified group instance.

Provision is included for giving a program exclusive or protected retrieval or exclusive or protected update rights over one or more areas. No concurrent program can gain access to the area over which a program has acquired such rights. Exclusive control is obtained in an OPEN statement, and is relinquished when appropriate CLOSE statements are executed, or the program terminates.

The following table reflects conflicting (x) usage modes between two run units attempting to execute an open statement on the same area:

	EXCLUSIVE UPDATE	EXCLUSIVE RETRIEVAL	PROTECTED UPDATE	PROTECTED RETRIEVAL	UPDATE	RETRIEVAL	none
none	x	x	yes	yes	yes	yes	yes
RETRIEVAL	x	x	yes	yes	yes	yes	
UPDATE	x	x	x	x	yes		
PROTECTED RETRIEVAL	x	x	x	yes			
PROTECTED UPDATE	x	x	x				
EXCLUSIVE RETRIEVAL	x	x					
EXCLUSIVE UPDATE	x						

IDS

When a group schema is defined, an authority code number can be provided to inhibit all unauthorized attempts to STORE, MODIFY, or DELETE any such group. When issuing an OPEN statement the user can supply a number to be used as an authority key. The "AUTHORITY-KEY clause" enables access to various group types which may be protected by a defined "AUTHORITY code". The value of the authority code number may not exceed 4095. When this clause is used in the OPEN statement, each reference to a group of the file involves a match of the authority value defined for the group with the authority key supplied. When a valid match occurs, the statement is executed, otherwise, it is not and an error condition is returned.

Note that the data definition and the DML statements are in the same program. This means that the locks and keys to them are both defined within the same user program, that is, by the same person.

IMS

The system protects certain groups in a file through the concept of "sensitive" groups. A program can only reference those groups to which it is bound ("sensitive") as specified in the "PCB". In addition, any program can be restricted to read-only operations against its "sensitive" groups. Also, access can be restricted to named terminals.

Security clearance takes place on two levels. First a password can optionally be required to initiate execution of a program, and secondly, certain data is protected from the actions of each program by the data administrator who specifies those files and groups which the particular program can access.

SC-1

A program can only reference the data structures to which it is bound.

When a user initiates the running of a program he must give his name. Then when the OPEN statement is issued, the system looks at the "User Security List" maintained by the Data Administrator (see Chapter 8) to find out his "Security Restriction Level (SRL)" and his access or modification codes.

This is compared to the SRL and access/modification codes required to process the file to be opened. Since SRL and access or modification codes are specified at the level of individual data items, the user's authority to act on the data is checked each time a DML statement is issued.

When a file is opened for modification (OUTPUT or UPDATE), no other concurrent program is permitted to open any part of that file.

7.5 Data manipulation language statements

The actual DML statements available to the programming user may be classified in various ways. Four classes are identified as: control statements which do not result in any data movement, retrieval statements which move data from the data base to the user working area with no change in the data base, modification statements which cause the data base to be changed in some way, and special purpose statements which do something other than deal with a data base stored on secondary storage devices.

7.5.1 Control statements

Statements may be provided to control the processing of the data base. The programming user may use a pair of statements, open and close, to begin and end processing a portion of the data base. A statement may also be provided to control the sequence of statement execution within the host language program based upon some condition pertaining to the data base.

7.5.1.1 Open

Using an open statement the user signals his intention to begin processing a portion of the data base. He may be required to identify what part of the data base is to be processed and in what mode the processing is to be carried out.

The system then responds by ensuring that the requested portion of the data base exists and is available to the program for processing, by creating any tables and user working areas to be used during processing, and perhaps by performing other related activities such as security clearance, lockout, and binding. Generally, a data item cannot be referenced until it is included within the scope of an executed open statement.

If all data in the system is available to the program during execution or if the extent of the data base is limited externally, by permitting only one file to be handled by the system at once, then the open/close control statements are not necessary.

COBOL

A file is opened with the statement:

```
       ┌                                      ┐
       │ INPUT   file-name       [REVERSED ]  │
       │                         [NO REWIND]  │
  OPEN < OUTPUT  file-name       [ NO REWIND ] > ...
       │                                      │
       │ I-0     file-name                    │
       └                                      ┘
```

A file named in an open statement can be designated as INPUT, OUTPUT, or I-0. The I-0 (input-output) mode can only apply to direct access storage files since both reading and writing are allowed only with such files.

Label checking is performed by the open facility if specified in the File Section of the Data Division. Additional label checking procedures can be defined with the USE statement:

```
      ┌      ┐            ┌           ┐  ┌    ┐
      │BEFORE│            │BEGINNING* │  │REEL│
  USE <      >  STANDARD  │           │  │FILE│
      │AFTER │            │ENDING*    │  │UNIT│
      └      ┘            └           ┘  └    ┘

                           ┌ file-name-1  [, file-name-2]... ┐
                           │ INPUT                           │
  LABEL  PROCEDURE  ON    <  OUTPUT                           >.
                           │ I-0                             │
                           └                                 ┘
```

This statement is followed by the desired subprogram statements. If the words BEGINNING or ENDING are not included, the designated procedures are executed for both beginning and ending labels.

For INPUT or OUTPUT files if the external storage medium permits rewinding, either the REVERSED or NO REWIND options may be specified. With no option specified the file will be positioned at the beginning by the open statement. With NO REWIND, the system assumes that the file is correctly positioned at its beginning and open does not reposition the file. With the REVERSED option, the file is assumed to be correctly positioned at its end, in which case subsequent READ statements make file entries available to the program in reverse order.

If an INPUT file is designated as OPTIONAL in the File Control paragraph of the Environment Division, the OPEN will be executed and the first READ of the file will return an end-of-file condition if the file is not present.

Of a set of files, which are defined in the Environment Division to share the same system buffer (area), only one can be opened at a time.

DBTG

The statement:

$$\text{OPEN} \left\{ \begin{array}{l} \text{ALL} \left[\text{FOR SET relation-name-1 [,relation-name-2]...} \right] \\ \text{AREA area-name-1 [, area-name-2]...} \end{array} \right\}$$

$$\left[\text{USAGE-MODE is} \left[\begin{array}{l} \text{EXCLUSIVE} \\ \text{PROTECTED} \end{array} \right] \left\{ \begin{array}{l} \text{RETRIEVAL*} \\ \text{UPDATE} \end{array} \right\} \right]$$

is used to specify the processing mode of one or more group relations or "areas" to be opened. Further program execution is delayed until such usage can be permitted, security clearance procedures are carried out, and "areas" are available, that is, on-line.

With the FOR SET clause, the open refers to the "areas" included in the auxiliary data definition that contain groups which participate in the named group relations.

The USAGE-MODE clause indicates the processing mode (see 7.2.1) and the integrity provisions (see 7.4.5) to be related to the opened "areas" or group relations. The EXCLUSIVE option prohibits concurrent program execution in any mode. The PROTECTED option prevents concurrent update and allows concurrent retrieval. Concurrent update is possible without the specification of "exclusive" or "protected". Note that the KEEP statement still provides the mechanism to preserve the integrity of the data base. The default mode of retrieval does not permit the use of any modification statements.

IDS

The open statement:

$$\text{OPEN} \left[\text{FOR} \left\{ \begin{array}{l} \text{RETRIEVAL} \\ \text{UPDATE*} \end{array} \right\} \right] \text{ [WITH AUTHORITY-KEY integer]}$$

is used to initialize processing of a data file. When the file is opened for input only (RETRIEVAL), the STORE, DELETE, and MODIFY statements are not operative. In the absence of the "FOR clause", UPDATE is assumed.

A run unit can only open one file during its entire execution. It is named in the Environment Division in a SELECT FILE statement.

If a group schema in the file to be opened is protected by an authority lock, the user must provide an authority key integer with the open statement in order to gain access to the group. The integer must match the one provided for the group in data definition.

IMS

The opening of a file for subsequent processing is implicit in the first reference to a file from the user program. Through the predefined "Program Specification Block (PSB)" the system sets up internal tables for each file referenced by the program at the time the program is loaded into primary memory. By comparison, other systems do this when the file is opened. The copy of the "language interface" control module which is loaded to process the first DML call statement is used throughout the execution of the run unit.

SC-1

The statement:

```
                            ┌                          ┐
                            │               OUTPUT     │
OPEN   file-name   FOR     <  [ RANDOM ]    INPUT       >  [cond].
                            │ [ RANDOM ]    UPDATE     │
                            └               UPDATE COPY┘
```

creates an internal table used to record the status of the opened file and a list of all the data items and groups to be referenced in that file. It then reserves the system buffer ("segment buffer"), issues the open through the operating system, and retrieves the first relevant physical storage record ("segment").

"File-name" refers to a particular auxiliary data definition, where it is associated with a data base file name. A planned feature of the system provides for the auxiliary data definition to contain a conditional expression which will select a subset of entries within the file.

When opening for random input the data may reside on either a sequential or a direct access device, although processing the former may be quite inefficient. With a random update open the data must reside on a direct access device. The output option is only used when no entries exist for that file. It signals the user's intention to only create a file in the data base.

The UPDATE COPY option has no meaning with data stored on a sequential device since a new copy of that portion of the data base will always be created during the update processing. If the data is stored on a direct access device and the COPY option is not selected, the opened file is updated in place and no copy of the old file is produced. With the COPY option, a copy of the new file is made either on tape or disk during the update. The disposition of the copy is determined by the Data Administrator through the "Data Location Definition" function (see Chapter 8).

7.5.1.2 Close

When the user is finished processing he may issue a close statement which signals the system to clean up the processing and release storage area used for tables. The open and close statements are generally required in pairs.

COBOL

A file is closed with the statement:

```
        ⎧             ⎡REEL⎤  ⎡NO REWIND⎤ ⎫
CLOSE   ⎨  file-name  ⎣UNIT⎦  ⎣  LOCK   ⎦ ⎬  ...
        ⎩                                 ⎭
```

The words REEL and NO REWIND apply only to sequential access devices; the word UNIT applies only to direct access storage files which are being processed sequentially. With REEL or UNIT words the close statement only acts on the current reel/unit of a multi-reel/unit file. The LOCK option is used to inhibit subsequent open statements on the closed file or reel/unit. Where it is appropriate to the file, reel, or unit, a rewind will take place; that is, the reel or analogous device is positioned at the beginning of its content, for a CLOSE or a CLOSE with LOCK.

In addition to standard trailer label checking, the user may specify his own procedures with a USE statement (see 7.5.1.1). They can be executed before or after the standard procedures.

If a file described with the OPTIONAL clause is not present when opened and read, the standard end of file procedures are not performed by the CLOSE statement.

DBTG

The statement:

```
       ⎧ALL  ⎡FOR SET relation-name-1 [,relation-name-2]...⎤⎫
CLOSE  ⎨AREA ⎣ area-name-1 [, area-name-2]...              ⎦⎬
       ⎩                                                    ⎭
```

is used to release control over "areas" by relinquishing any EXCLUSIVE or PROTECTED rights and making the "areas" available to delayed requesting programs and by relinquishing any holds ("KEEPs") on groups within the specified "areas". The parameters of the statement have the same meaning as in the open statement.

IDS

When processing is completed, a CLOSE statement is issued to transfer all modified pages currently residing in the system buffers to the direct access storage device:

CLOSE

No automatic closing takes place when a program normally terminates. Therefore a close statement must be issued prior to any "stop run" statement.

The close statement also causes the system to produce certain statistics reports which are appended to the IDS execution report. For each primary entry subroutine, those called directly by the object program, the report provides:

- primary entry subroutine name
- times called
- number of reads
- number of writes

Other reports appended to the execution report cover "data base attributes" and "total I/O performed on the data base" (see Chapter 8).

IMS

The programming user does not issue an explicit close statement. A file is closed: (a) when the system is brought down, (b) when the "master terminal operator" calls for a dump or restore of that particular file, or (c) in the event that primary storage space has been exhausted, a file may be closed in order to allow the opening of another. A further reference to a file would once again open it.

SC-1

The statement:

CLOSE file-name

will, when necessary, write the final storage block into the data base, copy the balance of a data base area open for update copy, and then release primary storage.

7.5.1.3 Conditional

A statement which tests a conditional expression provides the programming user with the ability to modify the sequence in which host language statements are executed, that is, to take some action, based upon a condition which relates specifically to the state of the data base as opposed to its content. This is not to be confused with the conventional IF statement. Similar statements may be used to modify the processing sequence based upon the detection of an error condition (see 7.4.3).

COBOL

none

DBTG

The IF statement makes it possible to execute parts of his program based upon the results of evaluating a condition relating to the data base.

```
      ┌                                                            ┐
      │relation-name-1  SET  [NOT]  EMPTY                          │
   IF {                                                            } ;
      │RECORD  [NOT] ┌OWNER ┐ OF ┌relation-name-2┐  SET             │
      │              └MEMBER┘    └ANY            ┘                 │
      └                                                            ┘

      ┌statement-1  ┐    ┌        ┌statement-2  ┐┐
      └NEXT SENTENCE┘    │; ELSE  └NEXT SENTENCE┘│
                         └                       ┘
```

The possible conditions are: (1) whether or not the current group relation has any dependent groups, or (2) whether or not the current group is a dependent, a parent, or either, of any or of a named group relation. If a current group or group relation is not known, the condition is returned as false and the else clause is executed.

IDS

The programming user may alter the sequence of processing based upon the current group type. The format of the statement is:

```
IF  group-name  RECORD  statement-1  [ ; statement-2 ]...

  [ELSE   statement-3  [ ; statement-4 ]...]
```

The IF group-name clause supports the use of those retrieval statements where the type of group to be retrieved may not be known until after the retrieval is complete. It may only be used following retrieve direct, retrieve each, retrieve next, retrieve prior, or retrieve next of "CALC chain" statements.

If the group retrieved is of the type specified by group-name, statement-1 and subsequent statement-2's will be executed in sequence and then control will be transferred to the next sentence in the program. If the group retrieved is not of the type specified, then control will be transferred to the ELSE clause or to the next sentence in the absence of an ELSE clause.

Statements-1,2,3, and 4 may be any one of the following statements: MOVE TO WORKING STORAGE, MODIFY, DELETE, HEAD, unconditional transfer, or subroutine call. In addition, statement-3 may be another IF group-name clause. This allows multiple test branch logic based on group type.

IMS, SC-1

none

7.5.2 Data retrieval statements

Data retrieval statements are directed toward locating data and making it available to the program in the user working area. Retrieval statements never modify the contents of the data base.

7.5.2.1 Locate

Locate statements will cause the system to locate instances of entries, groups, or items in the data base but will not actually result in any transfer of data to the user working area.

In effect a pointer is established to the data in preparation for subsequent processing. The locate statement may be used to establish the absence of a group or to locate one or more groups. The search is usually based upon some type of criteria such as the position of the current group, or the satisfaction of given selection criteria.

COBOL

The statement:

```
SEEK  file-name
```

is used to initiate the accessing of an entry from direct access storage for subsequent reading or writing. The SEEK statement is only used in conjunction with the reading and writing of files declared for random access mode. With random access files, the system must first locate the position of the desired entry and then read or write the entry.

An entry identifier must be provided to the system by the program prior to execution of the SEEK statement. It is placed in the data item variable declared as ACTUAL KEY in the File Control paragraph of the Environment Division (see 7.2.2).

If the programming user provides multiple entry storage areas in the user working area then multiple SEEK statements can be issued for the same direct access storage file before any are processed by a read or write statement.

DBTG

The FIND statement:

FIND record-selection-expression [SUPPRESS-clause]

is used to locate a particular group specified by the "record-selection-expression". When the group is found all four currency pointers are updated to reflect this group unless suppressed. The suppress clause is used to selectively suppress the updating of one or more of the currency pointers except "current group of run unit" (see 7.5.2.5).

A successful FIND results in the "area" name of the selected group being placed in communication item AREA-NAME and the name of the group schema of the selected group being placed in RECORD-NAME.

If a FIND is unsuccessful, the data base and the user working area remain in the state existing prior to the attempted execution, and the error communication items are set. Even though a FIND is successfully executed a warning code may be returned in ERROR-STATUS to indicate that a REMOVED or DELETED group was involved in the selection of the current group. For example, the selection may have specified the PRIOR group of the current group, but that current group was subsequently deleted or removed from the group relation.

The "record-selection-expression" may be one of the following seven forms to identify the desired group:

(1) [group-name] USING item-name

Prior to execution the named item must have been initialized with the unique internal identifier of the desired group. The user can further qualify the selection by indicating that the desired group must correspond to the named group schema (type).

```
(2)                                                    ┌ group-name  RECORD    ┐
         [OWNER IN relation-name OF] CURRENT OF        │ relation-name  SET    │
                                                       │ area-name  AREA       │
                                                       └ RUN-UNIT              ┘
```

Without the OWNER option this selection expression selects the group that is current based upon one of the four currency pointers. If the current of a group relation is desired and the current group of the named relation was REMOVED as a result of a previous operation by the same or another run unit, since it became current in this run unit, the FIND is executed and a warning indicator is returned.

Use of the CURRENT OF RUN-UNIT permits revision of currency status indicators which were previously suppressed.

With the OWNER option, first a current group is selected as above, then its parent group in the named relation is selected.

```
        ┌ NEXT      ┐
        │ PRIOR     │
        │ FIRST     │                          ┌ relation-name  SET ┐
(3)     │ LAST      │  [group-name] RECORD OF  │                    │
        │ integer   │                          └ area-name  AREA    ┘
        │ item-name │
        └           ┘
```

This form is used to select a group from a named group relation or a named "area" based upon the current group, the first group or the last group. Position in an "area" is determined by the unique internal group identifier. Position within a group relation is determined by the logical order of the relation which is established by an ordering clause during data defintion. The FIRST, PRIOR, NEXT, and LAST options are interpreted in one of these two ways depending upon whether the statement refers to an "area" or a group relation.

If a group name is given only groups of that type will be considered. Also, only groups declared in the auxiliary data definition to be part of the group relation or the "area" are considered.

The integer or named item provide a non-zero integer which selects a group based upon its ordinal ranking relative to the beginning, if positive, or ending, if negative, of the named relation or "area".

(4) OWNER RECORD OF relation-name SET

This selects the parent group of the current group of the named relation. It is exactly equivalent to format (2) with both relations naming the same group relation.

(5) [NEXT DUPLICATE WITHIN] group-name RECORD

This form is used only to select groups that have been declared as "calculated" in the data definition. Prior to execution of the FIND, the items, from which the group identifier is calculated, must have been previously initialized.

Without the DUPLICATE option, the first group with the same value for the calculated identifier is selected.

(6) group-name VIA [CURRENT OF] relation-name

[USING item-name-1 [, item-name-2]...]

This form selects an instance of the named group within an instance of the named group relation. With the CURRENT option the group is selected from the current group relation. Otherwise the group relation is selected based on the "SET OCCURRENCE SELECTION" clause in the data definition. All data items named in that clause must be initialized prior to execution of the FIND.

With the USING clause items in the named group can be used as selection criteria. The selection will search for a group with all values equal (conjunctive).

(7) NEXT DUPLICATE WITHIN relation-name

USING item-name-1 [, item-name-2]...

This form causes a search of the dependent groups of the current of the named group relation for a group which is of the same type as the current group and which has the same values for the named items. All values in the user working area are ignored. The search is in the NEXT direction, starts from the current group of the named relation, and continues until a duplicate is found or the end of the group relation is reached.

IDS

```
The statement: ⎧ group-name RECORD                                 ⎫ (1)
               ⎪                                                   ⎪
               ⎪ CURRENT group-name RECORD                         ⎪ (2)
               ⎪                                                   ⎪
               ⎪ ⎧NEXT  ⎫                                          ⎪
               ⎪ ⎨PRIOR ⎬ RECORD OF relation-name CHAIN            ⎬ (3)
     RETRIEVE  ⎨ ⎩MASTER⎭                                          ⎪
               ⎪                                                   ⎪
               ⎪ EACH AT END GO TO procedure-name                  ⎪ (4)
               ⎪                                                   ⎪
               ⎪ DIRECT                                            ⎪ (5)
               ⎪                                                   ⎪
               ⎩ NEXT RECORD OF CALC CHAIN                         ⎭ (6)
```

causes the group referenced to be retrieved and placed in the system buffer. This action may or may not require that a block be transmitted from the direct access storage device, since the group may already be in primary memory. The group is not moved to the user working area.

The unique internal group identifier ("reference code") of the group retrieved is accessible in the communication item named "DIRECT-REFERENCE" after the retrieval process is completed. The group retrieved is recorded as the current group of its type and is the current group in each relation in which it is defined whether as parent or dependent. If a group cannot be retrieved according to the specifications of the retrieval statement, an error condition is noted.

Of the retrieve options available, (1) and (5) are absolute in the sense that the desired group can be identified independent of previous processing. The others are context dependent, since the actual group retrieved is dependent upon previous group processing.

(1) In the first option the retrieval action is accomplished by one of three methods predicated upon the "RETRIEVAL VIA..." clause of the group definition in the data definition.

The group is retrieved based upon the values placed in control data items in the user working area by the user prior to issuing the retrieve statement. If the group is to be retrieved via item name, the contents of the named item will provide the unique internal group identifier of the object group. If retrieval is via a named group relation, the contents of its "MATCH-KEY" and ascending and descending sort items are used. If retrieval is via calculated addresses, the items from which the randomized address is calculated are found in the user working area.

(2) The group retrieved will be the current group of the group type named, regardless of the processing that preceeded since it was established as the current group. If no group of this type has been processed, or if the last group processed of this type was deleted, an error condition is noted.

(3) In these cases, the retrieval depends upon the current group within the relation specified. If NEXT or PRIOR is used, the appropriate group is retrieved regardless of the group type. When MASTER is specified, the parent group of the relation named is retrieved. If no groups of the relation have been processed, or if the last group has been deleted, such that no groups exist in the group relation an error condition is noted.

(4) The retrieve EACH option facilitates a "reference code" ascending sequence serial search of the data base. This statement will retrieve the first group, in ascending unique identifier sequence, whose unique identifier is equal to or greater than that stored in the communication item named FIRST-REFERENCE. However, if that retrieved group's unique identifier is equal to, or greater than the value stored in the communication item named LAST-REFERENCE, control will be transferred to the procedure named. When a group is retrieved, the sum of its unique identifier plus one will be stored in FIRST-REFERENCE, initializing it for a subsequent execution of RETRIEVE EACH. The unique internal identifier for any group reflects its physical location on the direct access storage device. The high order bits give "page" number and the lower order bits give a "line" number within the physical "page". In data definition the user has some control over the assignment of unique internal identifiers to groups.

(5) For direct retrieval, the unique internal group identifier for the desired group is supplied in the communication item named DIRECT-REFERENCE prior to issuing the RETRIEVE statement.

(6) This option is used to retrieve the next group of a "calculated address" group relation regardless of the type of the current group. If a current group of the relation does not exist (established through previous processing of groups in the relation) an error condition is noted.

IMS

No statement is provided to accomplish location only. Location of data in the data base is done in conjunction with the access statements.

SC-1

The statement:

```
FIND {rgroup-name        IN  file-name    cond}
     {rgroup-name  EOF   IN  file-name        }
```

is used to position the internal currency pointer within a file opened for random processing. The search always begins at the start of the current instance of the file/assembly specified by rgroup-name. The currency pointer is either positioned at the first group which satisfies the selection criteria ("cond") or at the end of the file/assembly.

If UPDATE COPY is specified or implied in the open statement, the FIND statement will copy the data as it advances through the opened file.

The statement:

```
SEEK {rgroup-name        IN  file-name    cond}
     {rgroup-name  EOF   IN  file-name        }
```

is used to advance the currency pointer through the opened file, which can be opened for either sequential or random processing. In all other respects it functions like the FIND statement.

7.5.2.2 Locate and access

Access statements result in data being transferred from the data base into the user working area and made available to the calling program with no change in the data base. A single statement may be used to accomplish both the locating and accessing of data.

COBOL

The statement:

```
READ  file-name  [INTO identifier]  ;  {AT END     } statement
                                       {INVALID KEY}
```

is used to locate and access an entry of a random access file when no prior SEEK statement was executed. The READ becomes a simple access statement when a prior SEEK was executed or when the file is declared as sequential access.

For sequential processing, the READ statement makes available the next logical entry from an input or an input-output file on a sequential storage device or for files on a direct access storage device but processed in a sequential mode. The END specifies an imperative statement to be executed when an end-of-file condition is detected.

For files in random access mode, READ implicitly performs the function of the SEEK unless it was explicitly issued for this file prior to the READ. In either case, the seek activity requires the user to provide an entry identifier item. The READ statement makes available the specified entry from the named file. If no entry was found with the given entry identifier, the imperative statement after INVALID KEY is executed.

If the INTO phrase is specified, the current entry is moved from the user working area to the area specified by the identifier, a data item name which can be qualified or subscripted. The move takes place according to the rules of the MOVE statement (see section 7.4.2). The entry is then available in both locations. If the file contains variable size entries, INTO cannot be used. When the logical entries of a file are described with more than one description, they automatically share the same area in the user working area making only the current entry available.

If the end of a reel/unit is reached before the logical end of file, the READ statement will execute the standard end of reel/unit label procedures, swap input devices, execute beginning label checking procedures, and make available the first entry on the new reel/unit. If a procedure is specified for USE before or after standard reel or label processing it will be executed.

DBTG

No statement is provided to accomplish locate and access. The equivalent is achieved with a FIND followed by a GET.

IDS

The following statement is used to retrieve the parent group of a specified group relation and to move all of its data items to the user working area:

```
HEAD relation-name CHAIN
```

This is the only IDS statement which locates and includes an implied move of the group to the user working area. Some group within the named relation must have been previously processed. If no group of the group relation was processed, or if the last group has been deleted, an error condition is noted.

After execution of this statement, the parent group retrieved is the current group of its respective type. It also becomes the current group in each relation in which it is defined as a dependent. However, it does not become the current group in any relation in which it is defined as a parent group. In those relations, the current group remains unchanged. Note that the function of the statement is similar to that of the RETRIEVE parent group; MOVE TO WORKING-STORAGE, except for the manner in which the "chain tables" are updated, to reflect the current group of the relation.

If the dependent groups of the named relation were defined to contain a link to parent, the system can directly access the parent. Otherwise, it will scan through the remaining dependents of the group relation.

IMS

The retrieval of groups is accomplished by the three GET statements.

GET UNIQUE (GU) is used to locate and access a single group as defined by selection criteria. The selection criteria must uniquely identify a group at each level in the hierarchy down to the desired group.

At the top or entry level of the hierarchy the selection criteria must reference the item designated as the identifier ("key") item. Also, the relational operator may only be equal (=), equal to or greater than (=>) or greater than (>). If no identifier value is supplied for a group below the top level, the first instance of the named group is selected.

This statement moves forward or backward in the file in searching for the desired group. At lower level groups in the hierarchy, if the file is declared random, the search item may be any defined item. For sequential files, only the lowest level search item may be an item other than the identifier.

GET NEXT (GN) may be issued with or without selection criteria. In either case the search moves in strictly a forward fashion from the current position. If no current position has been established, the search proceeds from the beginning of the file.

With no selection criteria, the next group is accessed without regard to its schema (type) or level.

The selection criteria may begin at any level not just the "root" group, but then it must continue for each level down the hierarchy to the desired group. The selection criteria at a given level may simply name a group type or may specify a condition on an item of a named group type. If an item is named it must be the group identifier item for all levels except the bottom one in the search.

GET NEXT WITHIN PARENT (GNP) operates the same as a GET NEXT statement except that in its search the system will not go outside of the tree rooted by the current parent group. Each successfully executed GET UNIQUE or GET NEXT establishes the parent of the retrieved group as the current parent. This statement may be used to retrieve all groups or selected groups within the family of the current parent group. An end-of-file or not-found condition results when the system encounters a group that is at the same level as the parent or higher.

SC-1

The statement:

```
READ  group/item-name  IN  file-name [INTO rgroup-name] [cond]
```

is used to transfer data into the user working area.

The system always advances through the data base, even if the file was opened for random processing, and regardless of whether or not a complete group is being transferred. For example, if a group schema consists of two items, A and B, the statements READ A followed by READ B would access data from two successive groups. If update with copy is requested or implied, an entry or group that has been read is not copied until the next statement is executed which causes the currency pointer to advance.

Applying a conditional expression to a READ statement effectively combines the SEEK function with the READ function. In the current release the conditional expression can only apply to one level at a time. The ability to refer to multiple levels in the hierarchy with the conditional expression is planned.

Within a hierarchy of groups the user must advance down through the levels in sequence as he issues READ statements. However, he may skip levels in backing up through the hierarchy.

The statement:

```
OBTAIN  group/item-name  IN file-name  [INTO rgroup-name]  cond
```

is used to transfer data items from a file opened for random processing to the user working area, with the search proceeding from the start of the named group or the group schema of the named item. The net effect of this statement is to combine a FIND followed by a READ with no conditional expression.

A third statement is used to read "flattened" files:

ACCESS file-name [UNTIL item-name]

To use this statement the auxiliary data definition for the named file can refer to, at most, only one repeating group at each level in the hierarchical structure of the data base. The user can view the single hierarchical path so defined as a single level or "flat" file. The groups appear in parallel in the user working area in the sequence which corresponds to their hierarchical sequence in the data base.

Each successive execution of an ACCESS statement (without UNTIL) reads the next group instance in the opened file, regardless of its level in the hierarchy, and leaves unchanged the contents of the user working area corresponding to all higher level groups in the hierarchy. The user must determine the portion of his working area that changed, that is, the repeating group schema that was read.

To use the ACCESS statement the named file can only be open for input. The only other operable statements are open, close, RECORD, and SET (see 7.5.2.5).

With the UNTIL option the ACCESS statement will skip repeating group instances until it encounters one containing the named data item. If a higher level repeating group must be passed in order to locate the next instance of the named data item, the data from the higher level group is read into the user working area. The ACCESS UNTIL statement, in effect, permits the user to read the next instance of a specified repeating group schema without regard to hierarchical boundaries.

7.5.2.3 Simple access

Simple access statements are used only to make data available in the user working area after location has been completely determined. Therefore, simple access statements never have selection criteria. They are oriented to moving items from the system buffer to the user working area. A system offering a general purpose locate and access statement does not need a simple access statement.

COBOL

none (the MOVE statement transfers data already available in the user working area to some other area of storage or visa versa).

DBTG

The GET statement is used to transfer data item values from the current group of the run unit, previously located with a FIND statement, into like-named data items in the user working area. The GET statement:

```
GET  ⎧[ group-name ]                                    ⎫
     ⎩group-name;  item-name-1  [ , item-name-2 ]...   ⎭
```

provides selective control over the data items made available. If the data items are defined differently in the data base definition and the auxiliary data definition then the system performs any necessary conversion. When no items are named all the data items in the group are transferred. Items are transferred from the current group of the named group schema. Without a named group, items are transferred from the current group of run unit.

If the GET statement cannot be successfully executed, the data base and the user working area remain in the state existing before the attempted execution. Missing data items in the group in the data base will result in null values being placed in corresponding items in the user working area.

IDS

The RETRIEVE statement only brings the desired group instance to a system buffer. The MOVE statement is required to make the data item values available to be referenced by other parts of the user program.

```
MOVE  TO  UWA  [item-name-1 [, item-name-2]...]
```

Values of the named data items, or all data items if none are named, are moved from the current group to the user working area. Only 02 level data items can be named in the MOVE. Each data item is unpacked into the user working area in accordance with the item descriptions provided in the data definition.

IMS, SC-1

none

7.5.2.4 Hold or reprocess

Some systems provide the ability to retain data in the user working area in anticipation of further processing or to lockout access by another run unit before this run unit is finished. The ability may be provided by an explicit statement which allows the user to say that he intends to process a group repeatedly and therefore the system should hold it until finally requested to release it. Alternatively, the ability may be provided through reprocess options in certain statements. Some systems may normally allow for reprocessing.

COBOL

No facility is provided to reread an entry. However, the INTO/FROM options are provided to obtain multiple copies of an entry when reading or writing.

DBTG

The statement:

KEEP

is used to advise the system of the programmer's intent to reprocess the current group of the run unit. Any later attempt to modify that same group will be successful only if another concurrent program has not altered the group with a MODIFY, DELETE, INSERT, or REMOVE statement since the KEEP was applied. In addition to the explicit KEEP statement, an implicit KEEP applies to all groups while they are the current group of the run unit.

The KEEP statement is not absolute; it merely signals intent. If the object group has been modified by a concurrent run unit, the current run unit may choose to modify the original copy of the group in any case, or it may reaccess the desired group to get the updated copy.

The statement:

FREE [ALL]

is used to cancel any KEEP which is in effect for the current group of run unit, or all groups to which an explicit KEEP currently applies in the run unit. Any KEEP is cancelled with a CLOSE or on termination of the run unit.

The KEEP and FREE statements operate within the OPEN declarations which may have specified PROTECTED or EXCLUSIVE.

IDS

none

IMS

Each of the three GET statements has a hold counterpart (GHU, GHN, GHNP) specifically assigned for update purposes. The hold option of a GET statement is used to identify the group to be modified. The hold function simply establishes the run unit's intent to modify the accessed group, which may or may not be exercised. The hold forms of GET permit the system to make certain that the group to be placed back in the data base is the same group that was accessed on the last GET HOLD statement.

SC-1

The THIS option can be used on the SEEK, READ, REPLACE or DELETE statements to reprocess the group last processed. A conditional expression cannot be used with the THIS option. The THIS option can only be used when the currency indicator points at a group (see 7.4.1), therefore, it may only follow a READ, OBTAIN, REPLACE or DELETE (but not delete file or repeating group instance).

SEEK THIS may only follow a READ or OBTAIN. It is normally used in the case where it is desired to add a new group in front of another group.

READ THIS steps back to the start of the last group processed and extracts (rereads) data from it. This allows repeated processing of a group.

DELETE THIS can reference an item or a repeating group, the latter case having a COMPLEX option (see 7.5.3.3).

7.5.2.5 Currency reset

Some systems provide special statements to reset the currency pointer to some location which was previously saved. Thus, an instance of data which was previously current can once again be designated as current.

COBOL

none

DBTG

The statement:

```
                           ┌ RUN UNIT                   ┐
                           │ group-name        RECORD   │
MOVE CURRENCY STATUS FOR  {  area-name         AREA      }  TO item-name
                           │ relation-name     SET      │
                           └                            ┘
```

is used to save the contents of one of the specified currency indicators. The system stores the unique internal group identifier ("data base key") for the current group in the named item.

A variation:

```
                      ┌                        ┐
                      │ RUN-UNIT               │
                      │ group-name      RECORD │
MOVE AREA-NAME FOR   {  area-name       AREA    }  TO item-name-2
                      │ relation-name   SET    │
                      │ item-name-1            │
                      └                        ┘
```

is used to save the name of the specified area in item-name-2. Item-name-1 is used to specify a unique internal group identifier.

The currency status indictors thus saved can be used in subsequent FIND statements.

The SUPPRESS clause may be used with the FIND or STORE statements to suppress the establishment of the object group as the current in one or more of the currency indicators.

```
          ┌                                                   ┐
          │ ALL                                               │
          │ ││ RECORD                                      ││ │
SUPPRESS {  ││ AREA                                        ││  }  CURRENCY
          │ ││ ┌SET                                     ┐  ││ │   UPDATES
          │ ││ {relation-name-1 [, relation-name-2]... }  ││ │
          └                                                   ┘
```

It cannot be used to suppress updating of current of run unit.

IDS

An option in the RETRIEVE statement is used to reset the currency pointers to the current (last processed) group of a particular group schema. With another option the programming user can place the unique internal identifier of the desired group in the communication item "Direct Reference" and issue a RETRIEVE DIRECT statement to reset the currency pointers.

The statement:

```
                          ┌               ┐
                          │ CHAIN-TABLE   │
                          │ MASTER        │
     MOVE  relation-name  { PRIOR         }  TO  item-name
                          │ CURRENT       │
                          │ NEXT          │
                          └               ┘
```

is used to save all or part of the "chain table" of the named group relation. The named item must be of the form PICTURE 9 (6) for each item of the "chain table".

IMS

none

SC-1

The statement:

```
RECORD  file-name
```

will store the unique internal identifier (IPC) of the data structure where the system is currently pointing in the named file.

The statement:

```
SET  file-name
```

will reset the system so that it points to (makes current) a data structure whose unique internal identifier was previously saved using a RECORD statement.

If no structure corresponds to the given internal identifier, the system positions itself where it would otherwise have been had the structure existed. The system issues a warning message under this situation. The user may choose to use this facility to establish a point at which to begin processing.

Only one pointer per opened file is maintained so that only one data structure is classed as current at any time. The user is required to provide the address of a storage space in the second parameter ("buffer address") of the CALL to DSKERNEL to store the unique internal identifier. If the user desires to save more than one current pointer in the same opened file, he must provide multiple locations in which to store the unique internal identifiers.

The file must have been opened for RANDOM processing to use these statements.

The programming user must be aware of possible adverse situations arising. For example, the unique internal identifier could change subsequent to a RECORD, if higher level groups are written which cause internal identifiers to "ripple" or overflow. Subsequent use of SET is meaningless. It is difficult but possible for the programming user to determine whether or not a "ripple" takes place.

7.5.3 Data modification statements

Modification statements include all those which can result in effecting a change in the contents of the data base. The new data is placed into the user working area before issuing the statement. Such statements may be provided to:

- add new data,
- modify existing data by replacing the previous contents with new data, or
- delete data currently stored.

Other statements may be provided which reorganize or reorder the data base without changing its contents.

7.5.3.1 Add

Certain modification statements may be used to add new data to the data base.

The add statement is usually provided to add data to the end of a file, to insert data into a file, or to populate a null file. The latter case enables initial file creation (see Chapter 6).

Some systems permit the user's program to be bound to some but not all of the items in a group. In this case, the system must have some policy for handling the unbound items when executing a user request to create a new entry. Often the system will store null values in unbound items. Alternatively, the system may require the user to be bound to all items in a group.

COBOL

All modification of the data base is accomplished through the use or disuse of the WRITE statement:

```
WRITE  entry-name  [FROM  identifier] ;

               INVALID  KEY  imperative-statement
```

It causes an entry to be added or inserted into an output or input-output file.

For direct access storage files in the sequential access mode, the imperative statement in the INVALID KEY clause is executed when the end of the last segment of the file (specified in the FILE LIMITS clause or in the ASSIGN clause of the Environment Division) is reached and an attempt is made to WRITE another entry.

In random access mode, the WRITE statement implicitly performs a SEEK unless one was explicitly issued prior to the WRITE for this file. The seek action requires that the user supply an entry identifier ("key"). If it is found to be INVALID, the imperative statement is executed and the entry in the user working area is still available.

After successful execution of a WRITE statement the entry is no longer available to the program unless the associated file is named in a SAME RECORD AREA clause.

If the end of reel/unit is detected on a sequential access, multireel/unit file, WRITE will swap output devices and do any label processing required on both the old and the new. The user may additionally provide label processing procedures.

If the FROM phrase is specified, the data is moved (non-destructively) from the area specified by the identifier into the user working area according to the rules specified for the MOVE statement without the CORRESPONDING phrase.

Additional clauses are provided with the WRITE statement to control spacing when the output file is intended to be written on a printer.

DBTG

The statement:

STORE group-name [SUPPRESS-clause]

is used to create a new group in the data base, using the data item values previously placed in the user working area. The new group is inserted into all group relations for which it is defined as a MANDATORY or an OPTIONAL AUTOMATIC dependent group according to any defined ordering. It is also added to each group relation for which it is defined as a parent. The group is established as the current of the run unit, and, if not suppressed, of the area in which it is stored, of the group type, and of all relations for which it is inserted. The communication items of AREA-NAME, and RECORD-NAME are appropriately updated.

The data items included in the data base definition but not included in the auxiliary data definition are supplied with null values. If item types are different in data definition and auxiliary data definition, the system will undertake the appropriate conversion.

All required initialization must have been done prior to execution of a STORE statement. This includes communication items as specified in the "Set Occurrence Selection" clause of each group relation involved, as specified in the "location mode" clause in the group definition, and subject to the constraints of the "WITHIN area-name" clause. For "Direct" groups the user may suggest a unique internal identifier to they system but it will not necessarily be used. Also, for the group to be properly inserted as a dependent in a group relation, the proper parent group must be established a current in each relation.

<u>IDS</u>

The STORE statement is used to place a group into the data base and to include it in any defined group relations. It can be used to populate an empty file, thus accomplishing initial file creation (see Chapter 6).

```
STORE   group-name   RECORD
```

When this statement is used, it is assumed that all items for the group have been initialized with the desired values in the user working area, and that any other control data items required to provide unique identification of all the parent groups of the relations which contain this group have been initialized in the user working area.

The group is placed into the data base in accordance with the clauses of the group description entry in the data definition. The group is stored in a physical block ("page") which is in some sense "close" to other groups of the same type or to its parent group depending on the user specifications. For groups stored according to calculated addresses, if the block determined by the randomizing data items is full, the group is placed on the first block with space available in the direction of ascending block numbers.

The group stored is recorded as the current instance of its type and as the current group in each relation in which it is defined. The unique internal identifier assigned to the group stored is accessible in the communication item named DIRECT-REFERENCE after the storage process is completed. The type of the current group is also available in the communication item named RECORD-TYPE.

If the storage process would create a group with identical values of identifer items when the group description specifies that duplicates are not allowed, the storage process is terminated and an error condition is noted.

When an entry (primary group) is stored, its unique internal identifier is moved to the item in user working area named by the RETRIEVAL VIA clause.

IMS

The INSERT (ISRT) statement is used to add a group to the end of a file declared for output (which may initially be empty) or to insert a group into a file declared for update.

Location of the position for the new group is specified in selection criteria which must uniquely identify a group at each level in the hierarchy from the "root" to the parent of the group to be inserted. The insert is performed so as to preserve ordering and uniqueness of group identifiers, if either of these have been defined.

Execution of the INSERT statement depends upon the rules specified for the group schema during data defintion. The clause:

$$\text{RULES} = (\begin{bmatrix} \text{P} & \text{P} & \text{P} \\ \text{L*} & \text{L*} & \text{L*} \\ \text{V} & \text{V} & \text{V} \end{bmatrix} \qquad \begin{bmatrix} ,\ \text{FIRST} \\ ,\ \text{LAST*} \\ ,\ \text{HERE} \end{bmatrix} \ )$$

specifies two types of rules. The first specifies which of three rules, designated as "physical (P)", "logical (L)", or "virtual (V)", are to be used for each of the three operations of INSERT, DELETE, and REPLACE.

Groups participating in a group relation are designated as "physical" or "logical." The (P) rule for INSERT specifies that a parent group of a group relation can only be inserted by its "physical" parent path. If the group is a "logical" dependent both its "physical" parent and its "logical" parent must already exist in the data base before the INSERT is executed.

The (L) rule specifies that a parent group can be inserted from "either side" of the group relation. If a "logical" (or "physical") parent and a dependent group of a group relation are to be inserted as one concatenated group under a "physical" (or "logical") parent, the insert is allowed. If the parent of the concatenated group already exists in the data base, it remains unchanged, and the dependent group is inserted.

The (V) rule is the same as the (L) rule except that if the parent group already exists in the data base, it will be replaced by the group in the user working area.

The second type of rule specifies how a group is to be inserted into an assembly. If no identifier is declared for a group schema, new groups are inserted as the FIRST or the LAST instance in the assembly. If a non-unique identifer item is declared, the new group is inserted as the FIRST or the LAST of the set of groups having the same value for the identifier.

HERE specifies that the new group will be inserted where the user establishes the currency pointer. If a non-unique identifier is declared and the user does not position the currency pointer among the set of groups with the same identifier value, the system inserts as the first of the set, thus preserving order.

SC-1

The statement:

```
WRITE  rgroup-name  IN  file-name  [ FROM rgroup-name ]
```

is used to transfer data from the user working area to the date base. The result will be to add a new group if open for output, or to insert a new group into a file if open for update. End of file marks are written automatically when the processing proceeds from one group schema to another or one level up to another in the data hierarchy. The location to add or insert the data must be found by the programmer issuing SEEK or FIND statements.

The programming user is required to provide values for all items in the groups to be written. For those items not bound to the program, the system automatically supplies null values or zero length.

7.5.3.2 Change

Change statements are used to change the item values within instances of entries and groups which previously existed within the data base.

COBOL

No explicit change statement is provided. Change is accomplished by reading an entry into the user working area, changing it in the program, then writing the changed entry to the data base. For direct access storage files, a WRITE with an existing entry identifier ("key") will overwrite the previous instance of that entry.

DBTG

The statement:

```
MODIFY   ┌ [group-name]                                      ┐
         └ group-name ; item-name-1  [, item-name-2]...      ┘
             [USING  item-name-3  [, item-name-4]...]
```

is used to replace the contents of all or named items in a group in the data base with values from like-named data items in the user working area. The affected group is the current of run unit. Naming the group serves as a credibility check on the type of the current group of the run unit. The values of those items excluded or not in the auxiliary data definition remain unchanged.

Data items which are defined to be used for parent selection, dependent ordering, calculation of a unique internal identifier, or constructing an index to search on dependent groups, may be modified. The system makes any required modifications in group relation parent-dependent relationships or in dependent group ordering to maintain the data base in accordance with the data definition.

The MODIFY statement is not executed if it would violate any "duplicates not allowed" clause. The MODIFY is also not executed if the current group has been modified by a concurrent run unit since the object group became current for this run unit or since a KEEP was issued on the object group.

IDS

To modify the contents of all or selected items of the current group and to relink any "chain" which may be affected by the modification of an "identifier" item, the following statement is used:

```
MODIFY ┌ item-name-1  [, item-name-2]...
       └ CURRENT  group-name  [item-name-1 [, item-name-2]...] }
```

The "current group-name" option is used to specify that the current group is to be modified only if it is of the type named. Otherwise, no modification takes place and an error condition is returned. Under this option, if no items are named, all items in the group are modified.

When an ordering item is modified, the group is relinked in accordance with the new value of its identifier item. When modifying a randomizing item, the dependent group is relinked into its relation according to the new value. When the item modified is a MATCH-KEY item, the association of the current group with its parent is changed. The group is delinked from its current parent, its new parent is retrieved, and the group is linked to its new parent according to the ORDER clause for the relation. The group then is a dependent to a different parent. In no case is a group ever physically moved from one relation to another. Consequently, an attempt to modify the identifier item of a "primary" group results in an error condition.

If the successful execution of the MODIFY statement would create DUPLICATE groups in relations where they are not allowed, the modification will not be executed.

IMS

The REPLACE (REPL) statement is used to modify data in an existing group. The group to be modified and replaced must first be obtained by a successful GET HOLD statement. The REPLACE must be the next DML statement after the GET HOLD. Between the GET HOLD and the REPLACE all desired modifications are made to the group in the user working area.

The group identifier item must not be changed.

The (P) rule (see 7.5.3.1) for REPLACE specifies that the group can only be changed by its "physical" parent path. The (L) rule is the same as (P) except that the run unit is not notified if the rule is violated. Under the (V) rule the group can be changed from either its "logical" or "physical" path.

SC-1

The replace statement to modify values in existing groups:

```
REPLACE  group/item-name  IN file-name [FROM rgroup-name]  [cond]
```

always advances through the data. The modified data is not output from the system buffer to secondary storage until the next statement advances the data pointer. This allows the user to issue REPLACE THIS statements on the data (see 7.5.2.4), causing the system to step back the data pointer to the start of the last group processed and modify it with the contents of the user working area.

The REPLACE statement can be used to change values of data items which are specified as ordering items. The system does not check to ensure that an ordering is maintained.

7.5.3.3 Delete

Most systems provide statements to delete data from the data base. Systems that permit some sort of parent-dependent relationship in the data structure may take various approaches to handling the dependent data when a statement is issued to delete the parent.

COBOL

No explicit statement is provided. For sequential files, not issuing a WRITE will effectively delete that instance from the output file. For direct access storage files, the user can write null data or establish his own delete indicator as an additional item in the entry. (This is currently under review by the CODASYL Programming Language Committee.)

DBTG

The statement:

```
                                  ┌         ┐
                                  │ONLY     │
    DELETE    [group-name]        │SELECTIVE│
                                  │ALL      │
                                  └         ┘
```

is used to delete the current group of run unit from the data base. The group is removed from group relations in which it participates based upon the qualification options:

No qualification - the group is only deleted if it has no dependents

ONLY -deletes the group and any MANDATORY dependents recursively, and removes but does not delete OPTIONAL dependents from the group relation.

SELECTIVE -as ONLY but also deletes OPTIONAL dependents recursively which are not dependents on any other group relation.

ALL -deletes all dependent groups recursively (at lower "levels") whether MANDATORY or OPTIONAL dependent groups.

As a result of a successful deletion the current of run unit is set to null (note that NEXT , PRIOR, etc. are still accessible).

Delete is not executed if the current group has been modified by a concurrent run unit since the object group become current or since a KEEP was issued on the object group. Delete is also not executed if the current group is not of the named type.

IDS

The statement:

```
DELETE   [CURRENT  group-name-1  RECORD]

         [ON-clause-1]

         [ELSE  ON-clause-2]...
```

Where the ON-clause is defined as:

```
ON  group-name-2  DETAIL

      [MOVE  TO UWA]

      [HEAD  relation-name  CHAIN]

      [PERFORM  procedure-name-1]

      [GO  TO  procedure-name-2]
```

is used to delete the current group and remove it from all relations in which it is a dependent, to delete all of its dependent groups, and, optionally, to perform certain functions when specified dependent group types are accessed during the deletion process. The execution of a delete statement makes the current group unavailable for further processing. Any subsequent attempt to access such a group results in an error condition.

If a group to be deleted is a parent with existing dependent groups, the system deletes all such dependents, beginning at the lowest level in the structure, before deleting the parent under consideration.

The "CURRENT group-name-1 RECORD" option is used to specify that the current group is to be deleted only if it is of the type named. Otherwise no deletion takes place and an error code is returned to the run unit.

The "ON-clause" is used to interrupt the deletion process each time a group of the type named is accessed. Under this condition, the various statements immediately following are executed prior to the deletion of the object group. After the execution of these statements, the deletion process continues, unless a GO TO statement was present in which case control tranfers to the named procedure.

The deletion process can be interrupted when more than one group type is accessed by using additional ON-clauses. When a dependent group is accessed and its type is not specified in an ON-clause, it is deleted normally.

The MOVE TO UWA statement under deletion control does not permit selective moves of data items (see 7.5.2.3).

If a PRIOR link exists in the deleted dependent group, the storage space is immediately available. Otherwise a flag is set so that the next time the dependent "chain" is traversed, the system can delink the previously deleted group and make the storage space available.

IMS

With DELETE (DLET) the specified group is logically deleted from the file including all of its dependent groups at all lower levels depending upon the delete rules for it and its dependents. The group is identified by a prior GET HOLD statement and the delete must be the next action on the file.

Under the (P) rule (see 7.5.3.1) the group can only be deleted by its "physical" parent path. If a "logical" parent group participates on one or more group relations, it may not be deleted until all "logical" dependents are deleted through their "logical" parent path. If the defined group is a dependent group, it must be deleted first from its "logical" parent path and then from its "physical" parent path. The (P) delete rule is propagated up the hierarchical path of the group defined with a (P) delete rule.

The (L) rule specifies that a "logical" parent will remain active when all "logical" dependents are deleted. It will be removed from the data base when deleted from its "physical" path. A "logical" dependent group can be deleted from either its "logical" or "physical" parent's path. However, the group remains in the data base and may be accessed through its non-deleted path. The group is removed from the data base only when it has been deleted by both its "physical" and "logical" parent paths.

Under the (V) rule a parent is deleted when it has no "logical" dependents or its "logical" dependents have delete rules of (V). With any "logical" dependents without the (V) rule, the parent is marked deleted on its "physical" path and is removed from the data base only when its last "logical" dependent is deleted. If a dependent group is being defined, it can be deleted from either its "logical" or "physical" parent path.

SC-1

The statement:

$$\text{DELETE}\left\{\begin{array}{ll}\text{assembly/file-name} & \text{[COMPLEX]}\\ \text{rgroup-name} & \text{[COMPLEX]}\\ \text{item-name} & \end{array}\right\}\ \text{IN file-name [cond]}$$

is used to remove data from the data base. To delete a non-optional item, the entire group must be deleted.

The COMPLEX option is used as a credibility check on the user who attempts to delete a data structure which contains a repeating group. The request to delete a "file complex" or a "group complex" is rejected with an error message if COMPLEX is not specified.

7.5.3.4 Reorder

Statements may be provided to reorder the sequence of entires in a file or groups within an assembly.

COBOL

Within the Procedure Division of a program the user can include a section to accomplish a sort on a special sort file. The section can include special processing to be performed on each entry before and after the sort is executed. The program may contain any number of sorts, each of which may have its own independent processing procedures.

The user issues the statement:

```
                 [ { ASCENDING  }                                    ]
SORT file-name-1 [ {            } KEY item-name-1  [,item-name-2]... ] ...
                 [ { DESCENDING }                                    ]

     { INPUT PROCEDURE section-name-1  [THRU section-name-2] }
     { USING  file-name-2                                    }

     { OUTPUT PROCEDURE  section-name-3 [THRU section-name-4] }
     { GIVING  file-name-3                                    }
```

to initiate the sort process. If the USING and GIVING phrases are used, the SORT simply writes the entries one by one from file-name-2 into file-name-1, a specially defined sort file, sorts the sort file according to the hierarchy of item identifiers specified in the SORT statement and writes the entries into file-name-3. The items used as sort identifiers must appear in each "record description", must not be variable length items, and may not be repeating items (that is, contain, or be subordinate to entries that contain, an OCCURS clause).

If an input procedure is specified, it must contain at least one RELEASE statement to transfer entries to the sort file, and can contain any other kind of processing to be carried out on the entries before they are written to the sort file. Program transfers cannot take place between the input procedure and other parts of the user program.

The output procedure is similar except that a RETURN statement is used to read entries back from the sort file and perform any further processing of the entry.

The FROM and INTO phrases are used with the RELEASE and RETURN statements respectively to achieve a dual transfer of data using an implicit MOVE, just as is done with the WRITE and READ statements.

DBTG

The statement:

```
                                          ⎧
   ORDER relation-name SET [FOR RUN-UNIT] ⎨ [FOR  group-name]
                                          ⎩                      ⎫
 ⎧⎡ASCENDING ⎤            ║ RECORD-NAME                      ║⎫  ⎬
 ⎨⎣DESCENDING⎦ KEY IS     ║ DATABASE-KEY                     ║⎬...⎪...
 ⎩                        ║ item-name-1  [,item-name-2]...   ║⎭  ⎭
```

causes all, or specified dependent groups in the named relation to be logically reordered in accordance with the items specified. The current group determines the parent whose dependents will be ordered. The basis for the ordering is provided by group name, unique internal identifier, or a named set of sequencer items. If the FOR group-name clause is not used, then the data items must appear in all groups. If the FOR RUN-UNIT option is specified the reordering of the relation ("set") is local to the run unit and disappears on termination of the run unit. That is, the original ordering of the relation is re-established upon termination of the run unit.

IDS

The SORT statement is used to reorder the dependent groups in a relation. The COBOL SORT is used and all rules for its use must be observed.

The group description of the special sort-file must specify as its first items:

-prior reference item
-current reference item
-sort item 1
-sort item 2

.
.
.

The input procedure must:

- -RETRIEVE the groups to be sorted,
- -MOVE the PRIOR reference to the prior reference item.
- -MOVE the CURRENT reference to the item in the sort group
- -MOVE the required data items into the sort key items.
- -RELEASE the sort group

The using file-name option requires the named file to be of the described format. The groups must be equivalent to those produced by using the input procedure option. All groups must be present in the selected relation.

The group relation, from which the groups to be sorted are drawn, may contain multiple group types. If only one type of group is selected for sorting, the selected sorted groups will be placed immediately following the parent group. The remaining group types will retain their relative order in the relation after all of the selected sorted groups.

At the completion of SORT, the last group in the sort sequence will be current of the program, current of type, current of relation name, and its data items will be moved to the user working area. It will not be current in any other relation in which it participates.

The "giving relation-name" option is used to name the new relation formed from the sort-file of sorted groups. Then in the output procedure, the RETURN statement is used to relink the sorted groups and make them available one at a time. Each group is moved to the user working area, and is made current of program, current of type, and current of relation-name.

IMS

No facility is provided to the programming user or the Data Administrator.

SC-1

There is no DML statement available to the programming user to accomplish reordering.

The system provides an independent Restructure function which enables the Data Administrator to reorder any file in the data base. The Restructure function uses the sort software provided by the manufacturer with the operating system.

7.4.3.5 Reorganize

Certain statements may be provided to perform some form of reorganization of the data base directly from a program, such as the rearrangement of entries in files or of dependent groups in group relations. These statements do not result in a change in the content of the data base.

DBTG

The statement:

$$\text{INSERT [group-name] INTO} \left\{ \begin{array}{l} \text{relation-name-1 [, relation-name-2]...} \\ \text{ALL SETS} \end{array} \right\}$$

is used to make the current group of the run unit, a dependent group of all or named relations on which it is defined as OPTIONAL. The specific parent in each relation is determined by the current group of each relation. The inserted group becomes current on all affected relations. The group-name is used to ensure that the current group is of the type named. The INSERT statement is not executed if duplicate group identifiers would result in any relation where duplicates are not allowed.

The statement:

$$\text{REMOVE [group-name] FROM} \left\{ \begin{array}{l} \text{relation-name-1 [, relation-name-2]...} \\ \text{ALL SETS} \end{array} \right\}$$

operates in the same way as the INSERT statement to remove the current group of run unit from all or named relations, provided that the group is defined as an OPTIONAL dependent.

COBOL, IDS, IMS, SC-1

none

7.5.4 Special purpose statements

The programming facilities discussed so far have been concerned primarily with data stored on secondary storage devices. DML statements are provided to control and manipulate data stored on secondary storage devices and to transfer data between secondary storage and primary storage (high speed core memory) where it can be referenced by the host language program.

Some systems offer additional programming facilities for handling data in a communiciations environment or handling data wholly within primary storage. If data base management is only concerned with the control and manipulation of a data base stored on secondary storage, then such special purpose statements would not be considered part of the generalized data base management system. However, it can be argued that such facilities are logically a part of data base management.

7.5.4.1 Table handling

Table handling statements are used to manipulate replicating data structures which are wholly within primary storage at the time of statement execution. The replicating data structures could be groups of a repeating group schema. Such an assembly or 'subassembly' may be created by the host language program or it may be populated by retrieving data from the data base in secondary storage. It is even possible for the system to allow a single statement to retrieve multiple replications of a group schema.

COBOL

A table handling capability is provided. A table is defined in the data division by adding an OCCURS clause to an item or group definition. The user working area is then established to contain all replications of the item or group. When a READ statement is used to retrieve an entry from the data base, all replications of each repeating group are placed in the user working area.

The OCCURS clause takes the following form:

$$\text{OCCURS}\left\{\begin{array}{l}\text{integer-1 TIMES}\\ \text{integer-2 TO integer-3 TIMES}\ \left[\text{DEPENDING ON item-name-1}\right]\end{array}\right\}$$

$$\left[\left\{\begin{array}{l}\text{ASCENDING}\\ \text{DESCENDING}\end{array}\right\}\text{ KEY IS item-name-2 }\left[\text{, item-name-3}\right]\ldots\right]\ldots$$

$$\left[\text{INDEXED BY index-name-1 [, index-name-2]}\ldots\right]$$

The replicating group so defined must be of fixed size. It can be defined to occur a fixed number of times using integer-1 or a variable number of times using integer-2 and integer-3 to specify the minumim and maximum number of replications, respectively. The minimum may be zero. The DEPENDING option is used to refer to a count item which always reflects the number of replications. An ordering may optionally be defined on the table. The data items must be part of the repeating group and not replicate within each instance of the repeating group. The ordering items are listed in decreasing order of significance.

The use of an OCCURS clause on an item or group which is subordinate to a group defined with an OCCURS clause (that is, a repeating group) gives rise to a multidimensional table. For example:

```
02  BAKER;  OCCURS  20  TIMES;  ...
   03  CHARLIE;  ...
   03  DOG;  OCCURS  5  TIMES;  ...
     04  EASY;  ...
```

defines DOG as a structure of a two dimensional table. A maximum of three dimensions are permitted.

Program references to structures in a table must refer to a single replication. This is done through subscripting or "indexing". The subscript may be an integer or a named item containing an integer. A subscript of 1 refers to the first replication in the table.

When "indexing" is used, the INDEXED BY option appears in the OCCURS clause and names "index" items which are analogous to index registers. The "index" items are dependent upon the hardware and cannot be associated with any data hierarchy. When referencing replications in a table the "index" names are written after the repeating group name and enclosed in parentheses. Relative "indexing" is specified by adding or subtracting an integer, literal or named item from the "index" name. For example:

```
DOG(INDEX1 , INDEX2-3)
```

The contents of an "index" must be established with a SET statement prior to executing a reference to the table using the "index".

The statement:

$$\text{SET}\left\{\begin{array}{l}\text{index-name-1 [, index-name-2]...}\\ \text{item-name-1 [, item-name-2]...}\end{array}\right\}\left\{\begin{array}{l}\text{TO}\left\{\begin{array}{l}\text{index-name-3}\\ \text{item-name}\\ \text{literal}\end{array}\right\}\\ \left\{\begin{array}{l}\text{UP BY}\\ \text{DOWN BY}\end{array}\right\}\left\{\begin{array}{l}\text{item-name}\\ \text{literal}\end{array}\right\}\end{array}\right\}$$

is used to initialize, increment, or decrement one or more named "indexes" with the value of another "index", an item containing an integer, or a literal integer. The increment or decrement may be a negative integer. The SET statement also allows the contents of a named "index" to be stored in one or more named data items.

The SEARCH statement provides facilities, through its two options, for producing sequential and controlled (for example, binary) searches. In using the search statement the programming user may vary an associated "index" name or an associated item name. This statement also provides facilities for execution of imperative statements when certain conditions are true and includes an AT END phrase.

```
         ┌group-name  [VARYING ┌index-name┐ ]┐
SEARCH  <                      └item-name ┘   >
         └ALL  group-name                    ┘

  [; AT  END  imperative-statement]
  {; WHEN  condition ┌imperative-statement┐ }...
                     └NEXT SENTENCE       ┘
```

The first search format starts with the replication in the table specified by the current value of the index. Successive replications are examined until one is found which satisfies any one of the conditions specified or the end of the table is reached. When the search terminates, the WHEN or END clause is invoked, and the "index" name will refer to the found replication. The ALL format is used to perform a controlled search on the table; the method being left up to the system implementor. The definition of a condition corresponds to that used in Interrogation (see Chapter 4).

DBTG, IDS, IMS, SC-1

none, except for those facilities provided in the COBOL host language.

7.5.4.2 Communications

Some systems provide statements to transfer groups of data items, called messages or transactions, between the user working area and queues which are often associated with external terminal devices.

Communication facility statements may be considered part of the data base management system when two points are considered. Incoming transactions may be used to interrogate or update the data base. Secondly, the transaction queues in the communications system may be considered a part of the data base.

COBOL

The characteristics of the communication devices and their associated queues are defined in the "Communication Description Area" of the "Data Division". Queues can be constructed in a hierarchy and handled using various disciplines such as fifo, lifo, or some priority scheme.

The RECEIVE statement is used to transfer a transaction from the queue of a named device to the user working area Alternate action is taken if the queue is empty. The SEND statement is used to transfer a transaction from the user working area to the queue of a named device. The statement can specify in addition that an "end of message" signal be transferred (when the message is built up in pieces) or an "end of transmission" signal be transferred. These statements are analogous to the READ and WRITE statements respectively.

The DISABLE statement is used to stop the transmission of transactions between a named external communication device and its associated queue. The inhibition can apply to input, output or both. The ENABLE statement is used to permit the transmission of transactions between a named device and its queue in either or both modes. Both statements have an associated security lock to prevent indiscriminate use of the facility by a programming user who is not aware of the total network environment, and who may therefore disrupt systems functions by the untimely enabling or disabling of a queue.

The ACCEPT and DISPLAY statements are used to transmit data from and to, respectively, a low speed communication device. No queuing of messages takes place.

IMS

The transfer of transactions is accomplished using the same statements provided for data base manipulation. The file name refers to a transaction queue and is defined in the "Program Communication Block (PCB)".

A transaction can consist of multiple group ("segments") of items where each group is at most 136 characters. The GET UNIQUE (GU) statement is used to transfer the first group of a named transaction from its associated queue to the user working ara. Subsequent GET NEXT (GN) statements are used to retrieve the remaining groups of a transaction.

For programs that process multiple transaction types, the text of the input transaction must be examined to determine the current type. Exception condition codes are used to indicate end of transaction (last group) or end of queue (last transaction).

The INSERT statement is used to transfer data from the user working area to a queue of transactions. An INSERT is issued for each group making up a transaction. The enabling and disabling of terminals is done at system generation time and is not under the control of the programming user.

DBTG, IDS, SC-1

none, unless provided in the host language.

7.6 Facilities for system programmers

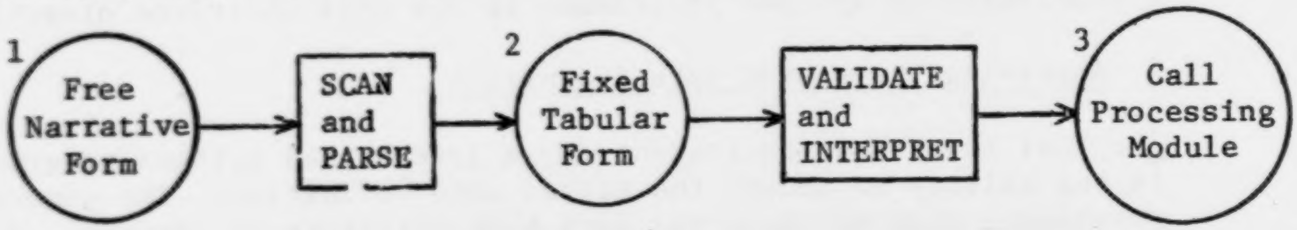
The data manipulation language statements of conventional programming facilities are intended primarily for the application programmer. He generally knows something about the logical structure of the data base which he is processing and high machine efficiency is often not an overwhelming consideration.

However, for the programming user who is writing system programs, such as an interrogation or update function, the need for generality and efficiency becomes more important - generality because he does not usually know the data structure in advance of writing the program yet the program must be able to deal with any kind of structure which is definable to the system; efficiency because system programs tend to be run more frequently.

The opportunity for greater efficiency can be possible by providing multiple levels of interface. Greater generality is possible when the programming user can access the stored data definition and deal with a general auxiliary data definition.

7.6.1 Multiple levels of interface

Some systems may provide more than one level of interface. In the following diagram, three possible interfaces can be identified:



First, the narrative or free form of writing the DML statements is most convenient for human users. This form also affords maximum opportunity for error checking during both the scanning phase and the interpretation phase.

The second interface is more like a machine interface using a fixed or tabular format. When a generalized program is written the programmer does not always know what DML statement is required, rather the program must be written so that during execution it will examine the situation and issue the appropriate statements. In such cases the fixed format is more convenient. It is also more efficient since less scanning is required.

The third interface identified does not involve any scanning, parsing, validation or interpretation of the DML statement. Such an interface will usually be reserved for advanced system programmers and throughly tested and debugged programs. The chief advantage of this interface lies in the efficiency that it affords in going directly to the processing module. This would likely be the interface used by the frequently-executed function modules such as interrogate and update.

From another point of view, it may be possible to classify the statements into different levels based upon the grossness or level of abstraction with which the data is referenced. Some statements may fall into a simpler, more primitive class while others may be a composite of the functions of more primitive statements. Some statements may reference data at the detailed physical I/O level, while others deal with data in terms of its logical structure.

None of the five systems offer a machine-oriented interface. None offer statements at the detailed physical I/O level.

IMS comes close by offering a separator rather than narrative form of statement.

SC-1 does have an internal table which is used to contain any DML statement in a standard way. The verbs and all their modifiers are coded into a few bytes. Although not encouraged, it is possible for a knowledgeable systems programmer to use this interface directly.

7.6.2 Accessing the stored data definition

The most important requirement for a generalized systems programmer is the ability to access the stored data definition. The generalized programmer does not know the data base structure in advance. He must write his program in such a way as to discover the structure being dealt with and then construct the appropriate DML statements. Often he will have no more than a file name and a few item names at some unknown level in the data hierarchy.

COBOL

There is no explicit provision for an auxiliary data definition which is different from the data definition. The definition of the data base is part of the program and hence the programming user knows the complete definition at the time the program is written.

DBTG

The language makes no explicit provision for accessing the stored data definition. It may be possible through an assembly language subprogram depending upon the implementation.

IDS

No access to the stored data definition. It is not required since a complete definition of the data base must be included in the Data Division of the host language program.

IMS

Access to the stored data definition by the programming user is not permitted. The programming user sees only the auxiliary data definition as specified in his "PCB".

SC-1

The data definition is stored in the data base and treated just like other data in the system. If a programming user has the required security clearance he can access any part of the stored data definition using the normal DML statements.

7.6.3 Handling a generalized auxiliary data definition

Sometimes a programming user has to write a program in a generalized fashion, for example, to develop a generalized interrogation capability for a non-programming user. In doing so, he is like a systems programmer. He is faced with the problem of referencing the data base and working areas without knowing the names of or relationships among the data elements he is dealing with in the data base structure.

If a programming user does not know the definition of the data in advance, he will also not be in a position to provide a complete auxiliary data definition and the user working area at the time of writing his program. To properly satisfy the situation, the system would have to allow the definition of the user working area to be deferred until program execution.

Some systems will operate with no items or only identifier items being defined in groups or entries. If the system assumes no definition of the remaining items, then the program must know the definition or obtain it elsewhere. This not not really aiding the generalized programmer but rather compounding his problem.

COBOL

Auxiliary data used by the program must be completely defined and correspond exactly to the data base definition. In fact, they are one and the same.

DBTG

The auxiliary data definition must be completely specified at time of program writing.

Various DML statements allow the user to name a group type which is then used as a credibility check on the type of the current group before the statement is executed.

IDS

The auxiliary data definition must be specified at time of program writing. It may be incomplete, thereby leaving the program the responsibility of knowing how to interpret items in the user working area.

IMS

For each group, only the identifier items need be defined. The program must use exactly the same definition as is in the stored data definition for all defined items. Undefined items are the responsibility of the program to interpret.

The user can issue DML statements without naming a group type. The name, level, and identifier item values of the current group are all accessible in the PCB after the statement execution.

SC-1

It is possible for the programming user to defer auxiliary data definition until program execution. A special copy option is provided at the time of auxiliary data definition, that is, "Bindlist" definition. When this is encountered by the execution of an open statement, the data base definition for the named file is copied and becomes the auxiliary data definition for the program. This facility was specifically designed as an aid to the generalized programmer.

8. DATA ADMINISTRATION FUNCTIONS

The role of an individual called the data administrator (sometimes data base administrator or data manager) varies considerably depending on the class of data management system and the capabilities it offers. When on-line capability is available, the role of the data administrator could frequently be merged with that of the systems administrator. However it is possible to identify a meaningful division of responsibility between these two, inasmuch as the systems administrator has a well identifiable role in the environment of any kind of time sharing or communications oriented operating system.

It is useful to identify two major differences between the role of the data administrator and that of the systems administrator. First the data structure and information content of the data base should have significant meaning to the data administrator, more so in fact than to any other individual in the enterprise.

The data administrator has the function of using his superior knowledge of the set of applications using the data base in order to maintain its integrity and security. In this sense he is initially responsible for all issues concerning data definition (see Chapter 3) and creation (see Chapter 6). Specific data administration functions are identified as

- assignment of pass keys
- specifying requirements for logging
- specifying an audit trail
- storage of permanent programs
- control over scheduling algorithm

To the systems administrator, the data base is merely a large quantity of stored representations, which as far as he knows may possibly be data, possibly programs, for which he has a full time responsibility. The systems administrator may even sit in the machine room, or at least have his own privileged monitoring console, so that he can control operations on a minute by minute basis. In case of difficulties he may invoke procedures, (possibly defined to him by a data administrator) to recover from breakdown and reconstruct (or restore) the data base. There is also a potential overlap between the role of the systems administrator and the machine room supervisor. In fact the clearer separation one identifies between the data administrator and the systems administrator, the closer the role of the latter merges into every day machine room supervision.

The role of a data administrator, in the sense discussed here, may be influenced by whether the data base management system functions in an on-line mode, or itself provides on-line capability. With no on-line capability, his job may be more oriented around the running of specific jobs such as file creation and the establishment and enforcement of people oriented procedures to control the integrity and use of the system. If on-line capability is provided, then protection of integrity becomes a system problem and the data base management system may itself provide special facilities for controlling the use of the system.

Restricting consideration of the data administrator to functions which call for an intimate knowledge of the data structure, it is possible to consider which specific functions can properly be ascribed to a single authority who may be one individual or a group of individuals depending on the size of the data base and the level of activity connected with it.

It is also of importance how the privileged functions for the data administrator are reserved for his use. In an on-line environment, a special password or a specially assigned console may be the technique chosen. In a batch processing environment the privileged use would normally be based on machine room procedures. In either case, some of his options may be exercised at the time the data base management system is generated; otherwise this happens later.

GIS

The functions of file creation (including data definition and security definition) are implicitly identified as reserved for use by what is referred to as "installation management."

MARK IV, NIPS/FFS, COBOL

No specific function is identified as restricted for use by a data administrator or other privileged user.

TDMS

No specific function is identified as restricted for use by a privileged user. However use of the ADEPT operating system calls for a"security officier" who is responsible for file security.

UL/1

The functions of file creation (including data definition) and file restructuring are explicitly identified as reserved for a data administrator. In the "Establishment Division", the data administrator must define the initial data structure and may ent any entry level validation criterion and a number of permanent computational procedures to be used either in the validation criterion or in subsequent interrogations and update.

In addition, the data administrator is the only individual allowed to use the "Revision Division" (see 3.9). With this capability, he may restructure the data file to allow new items or to delete existing areas from the schema.

The technique by which the data administrator functions are reserved for his exclusive use, relies completely on administrative procedures. In fact, any individual at an installation may use any part of the system.

DBTG

A data administrator is identified who is responsible for specifying the data structure or "schema" for the whole data base and possibly for specifying one or more "sub-schemas" for invocation by the user programs accessing a portion of the data base. He is ascribed three main functions, namely organizing, monitoring and reorganizing. Organizing is defined to include the assignment of data names, selection of search strategies, assignment of security requirements and the assignment of parts of the data base called "areas" to specific media types.

Monitoring the data base includes its usage, response, and breach of privacy. Logging and sampling are tools suggested for this purpose with a potential reorganization being the end result.

Reorganizing includes modification to the stored data definition, restructuring the data base to reflect changes in the stored data definition and initiating clean up or "garbage collection."

IDS

No specific function is identified as for use by a data administrator. However specification of the data definition would normally be performed by a single authority and capability is provided for defining sub-structures which are for use by applications programmers. The other data definition must be stored in the user programs.

IMS

Only one administrative function is identified for this system under the collective name of Systems Operation. Most of the functions can be identified as data administrator functions, although some would be typical for a systems administrator.

The function of preparing and entering the data definition is explicitly reserved for the data administrator. A program must be specially written to create the first instance of the data file. The data definition process is considered to subsume control of security, specification of storage media type, control over storage structure and the assignment of the names of groups which may be used in the primary data definition and in the applications. When non-hierarchic group relations exist, special utilities must be used in data definition.

The initiation of logging and the creation and update of the library file of transaction programs are both explicit functions for the data administrator. At systems generation time and at data definition time, he may exercise different levels of control over the scheduling algorithm. It may be adjusted within the prior constraints from a master terminal (see 8.1).

SC-1

The role of a data administrator is identified to include data definition, standardization of data names, restructuring of the data, control over storage media type, specification of security requirements, supervision of the auxiliary data definitions for programs and for update transactions, and specifications of the validation process. It is possible for the user of programming facilities (see Chapter 7) to augument the capabilities specifically provided.

8.1 Systems administrator functions

The precise role of the systems administrator with respect to a data base management system depends on the class and capabilities of the system itself and on the capabilities of the operating system under which the data base system is used. In one case, the data base management system may provide capabilities which, in another case, are an inherent function of the operating system.

Time sharing and communications oriented operating systems frequently provide many of the monitoring and support facilities which enable the system administrator to assume responsibility for the correct functioning of a set of programmed applications which are executing in a real time environment. Whether a data base management system is a part of this set of applications or not normally has no effect on the task of the system administrator. He should retain responsibility for allocating the physical resources of the machine, assuming a set of competing applications, one or more of which may use the data base management system. In this case an operating system component may monitor the use of the physical resources assigned and report to the systems administrator on the effectiveness with which they are being used. Typical facilities for use by a systems administrator include check point printing of the status of the data and the programs and the dumping of transaction queues at regular or at specific points in time.

In view of his close involvement with the running of the systems (operating and data base management), the systems administrator may have a privileged access via a specially dedicated terminal or a special pass word from any terminal. He may in fact perform as a highly trained operator sometimes called a "master terminal operator" and sit at a control console, watching displays or typeouts which give him information regarding the functioning of the system.

The tools available to the systems administrator to recognize system malfunction vary, and the tools provided to facilitate recovery from such malfunction can vary even more widely. The use of such tools and the data administrator's degree of involvement with their use are also matters which depend largely on the operating environment in which the data base management system is used (see Chapter 10).

GIS, MARK IV, NIPS/FFS, TDMS, UL/1, COBOL, DBTG, IDS

There is no explicit concept of a systems administrator associated with the data base management system. The capabilities which would be ascribed to such an individual are in all cases considered part of the use of the operating system.

IMS

A user environment may chose whether or not to be on-line. If so, it is necessary to have a master terminal. This may be the console typewriter ot it may be a remote terminal device. Normally a master terminal should be provided and its operator is regarded as being under supervision of the "systems operation function."

SC-1

A job definition function is provided to permit dynamic modification of the operating system's job stream. The execution of all programs including the self-contained functions always depends upon a prior execution of Job Definition to generate the required job control statements based on the stored storage structure definition. The output of Job Definition can be catalogued and later invoked by a job request.

8.2 Data administrator initiated processes affecting other users

The tasks of the data administrator may be divided into two main classes - those which he is responsible for, such as creation (see Chapter 6) and restructuring (see 3.9) and those in which he creates an environment for the other users of the data base. In the second class, he calls upon his superior knowledge of the data base and its continued usage to set parameters which constrain the various users, programmer and other, in the nature of the processes they may perform on the data base. Such processes could include assignment of pass keys, specification of logging and audit trail, and the selection of interrogation, updates and other programs to be stored permanently. Finally he might also control the actions which may be performed in such programs.

8.2.1 Assignment of pass keys

The declaration of privacy locks on any structural level in the data base is part of the data definition facility (see Chapter 3). The initial assignment and subsequent modifications of keys to open these locks are necessarily functions to be performed by the data administrator, whether he uses software or paper procedures. There is potential overlap in an on-line environment with the role of a systems administrator in assigning passwords to users which enable them to use the operating system and its associated language compilers from a terminal. However the operating systems password concept is potentially less complex than the data base management system privacy locks and keys, where a key may involve the creation of a procedure, possibly using the operating systems password as one of the input parameters.

GIS

Definition of security is performed with a separate utility task. A table of user security codes and corresponding data access codes is set up by the utility task. User procedures carry a security code which determines, through the table, the data which the procedure can access.

NIPS/FFS

A security lock code is appended to each file. A similar key code must be included in each interrogation. If they are different, a warning message is included in the output but the interrogation is completed. The maintenance of security is a function of the human administration.

TDMS

A security key may be centrally assigned to each user through the ADEPT operating system facilities. The file of such keys may be updated only from a security operations console. A periodic programmed check on the security status is also provided.

MARK IV, UL/1, COBOL

No capability is provided.

DBTG

A programmer must include a PRIVACY paragraph in the Identification Division of his COBOL program if the data base schema has privacy locks on any level. The features of the PRIVACY paragraph are considered programming facilities (see 7.4.5). Administration of the literal-names and procedure-names used as privacy keys is the responsibility of the data administrator.

IDS

Specification of security is essentially a programming facility (see 7.4.5) because the definition of the data structure for the data base and its locks and keys is all embedded within the program.

IMS

The data administrator may define pass words using a special security maintenance utility. When generating Program Communication Blocks (PCBs), he must define which program may access a file or part of a file. A list of programs and the files they may access must have been created at the time of systems generation. For each program in the list, the security maintenance utility is used to assign a pass word and may also include the mapping between physical and logical terminal names. The logical terminal name may have been previously defined during generation of the PCB for the given program. The programmer must use the pass word from one of the designated terminals at the time he invokes a program which accesses the data base.

SC-1

Data security is achieved by assigning security restriction levels, representing an "authority to know" to data items, and class codes representing a "need to know" to users. The user must satisfy both types of security restriction in order to access items in the data base. Every data item in the data base is assigned an authority level and a set of class codes for access and a set of class codes for update (see Chapter 3).

Similarly the data administrator assigns an authority level, a set of access codes and a set of modification class codes to every user in the system. A system function called User Security List Maintenance is provided for the exclusive use of the data administrator.

8.2.2 Specifying the logging of modifications to the data base

In a multi-user environment, logging of changes to the data base may be initiated either by the data administrator (based on his knowledge of the uncertain accuracy of the processes to be performed on the data) or by the systems administrator (based on his knowledge of the unreliable state of the hardware and/or operating system). If concurrent updating is allowed (see 10.2.1.2), then systems controlled logging is almost mandatory. If initiation of the logging is a programmer facility (see Chapter 7) then it would presumably be performed only for changes to the data base resulting from his program. Also it may be possible for the user of a self-contained update function to initiate logging (see Chapter 5). Alternatively logging may be a capability which is inherent in the operating system or in the data management system and difficult or impossible to suppress.

Logging is here identified as the process of recording, on a separate physical medium from the data base, any stored images which are changed during the processing of values in the data base. It is possible to have before image logging, after image logging, and before and after image logging. Normally the quantity of data logged corresponds to the physical block size transformed between secondary storage and high speed storage. The logging is performed specifically to facilitate restart in the case of system breakdown and the reconstruction of the data base to an earlier and presumably more correct status. It is a refinement of the practice of periodically dumping the whole file, which becomes less acceptable as the size of the data base increases.

The data administrator may have facilities at his disposal to require that logging be performed for a specific data base. The facility may consist of a privileged program which only he may use, which sets certain flags in the data base management systems control tables. Alternatively logging may be an operating systems function, in which case the data administrator may have to request the systems administrator to perform the required logging.

GIS, MARK IV, NIPS/FFS, TDMS, UL/1

There is no centralized control over the initiation of logging. Any that is provided is an inherent part of the update function (see Chapter 5) and/or an operating systems capability which may be invoked by control statements at run-time. Any facility for periodic dumping is normally provided by the operating system.

COBOL

No capability is provided.

DBTG

No explicit capabilities are identified for the function of logging. However it would be possible for the data administrator to define a procedure to be included in the data definition for the whole data base in the ON clause. This may be invoked at area, entry and group relation definition levels. This procedure could be used to log changes on the use of any programmer facility such as data modification statements (see 7.5.3) and/or possibly data retrieval statements (see 7.5.2).

IDS

Logging of both before and after images is an inherent part of the data base management system and is difficult to suppress. A "journal" file is produced which contains the time at which any modification to the data base occurs.

IMS

Logging of both before and after images is an inherent part of the data base management system. It may be totally suppressed at execution time for all parts of the data base, but not selectively. A "journal" file is produced which contains the times at which any modification to the data base occurs.

SC-1

Capability is provided for centrally controlled before and after image logging. In the update function (see Chapter 5), a user can cause logging transaction data after it has successfully been used to update the data base. The update transaction data definition includes specifications of a retention time period with a default of zero days.

8.2.3 Specifying the logging of transactions

In a multi-user environment, the logging of transactions entering the system from the terminals or the messages being sent by the system to the terminal can be of value to the data administrator in monitoring the kind of processing which is taking place during a period of time. If such a log file is generated, it is used by the systems administrator (or the master terminal operator) for recovery and reconstruction purposes. This logging is normally in the form of a machine readable file used optionally for recovery purposes or for statistical analysis.

GIS, NIPS/FFS, TDMS, IDS

No capability is provided.

COBOL

The communications capability is a programming facility (see 7.5.4.2). If facilities are provided for logging the flow of messages between the terminals and the control processor, then these would normally be part of the operating system.

IMS

Logging of both incoming and outgoint messages is always provided. Logging of queries can be suppressed, but not of updates. The log is in the form of a file which can be analyzed to derive performance and activity statistics.

MARK IV, UL/1, DBTG, SC-1

Not applicable.

8.2.4 Specifying an audit trail

An audit trail report differs from the file that is built up during the logging process in that the audit trail is specifically a report designed for study by the data administrator so that he can scrutinize the events that have taken place during a period of processing. Under certain circumstances, the log tape and the audit trail may contain essentially the same information in different form. However it is more likely that the log tape contain information regarding the physical resources of the hardware, whereas an audit trail would contain an indication of the events which have occurred to alter value content the data base (including possibly any of its directories) and would not necessarily be oriented toward hardware.

The generation of an audit trail could be initiated by the data administrator for a given data base, and/or for a specific period of time and/or under other specified conditions. This would then be a system oriented audit trail. As with the logging process, it is also conceivable than an audit trail could be initiated either by a programming user (see Chapter 7) or by the user of a self-contained function

GIS

The generation of an audit trail during UPDATE procedures is controlled through parameters supplied at data definition time. The user must redefine the data to change the parameters. Audit trails may also be produced by explicit user procedures.

MARK IV, NIPS/FFS, TDMS, UL/1

An audit trail can be generated only in a use of the update function (see Chapter 5).

COBOL

No capability is provided.

DBTG

A centrally initiated audit trail report may be provided in the same way as the logging of accesses to the data base (see 8.2.2), namely by use of a procedure defined by the data administrator as part of the data definition for the whole data base.

IDS

Specification of an audit trail is a programming facility (see Chapter 7).

IMS, SC-1

An audit trail can be provided only by processing the log tape (see 8.2.2).

8.2.5 Storage of programs acting on the data base

The concept of a library of programs which have been sufficiently tested to justify permanent storage on the available physical resources of the machine goes back to the batch processing monitor concept. In more recent time sharing operating systems, a terminal user can usually sorte his files (program or data) between terminal sessions on the system available storage space.

Within the environment of a data base management system, the importance of a library of frequently used and thoroughly tested programs is valid. A program in this latter sense is either a program in the conventional sense which uses programmer facilities (see Chapter 7) or a set of specifications to an interrogation function (see Chapter 4) or an update function (see Chapter 5) which has been prepared by a non-programmer. Determination of what is "thoroughly tested" and what is "frequently used" may be under control of the data administrator.

The preceding entities might be identified as transaction programs, application programs, or user programs, or some name defining the nature of the entity such as query or update. As part of the process of building up the library of frequently used programs acting on the data base, the data administrator may have the opportunity to assign an execution priority rating to each program. This rating would be taken into account when the program is being executed concurrently for other programs and is considered part of the operational environment (see Chapter 10).

GIS

Any kind of "task specifications" may be stored by the system in source form and procedural task specifications such as QUERY (see Chapter 4) may also be stored in object form. How this saving is done is a matter for control in the use of the system, rather than explicit privileged facilities being provided.

MARK IV

It is necessary to store transaction definitions in the system prior to a run to update the file (see Chapter 5). Frequently used information requests (see Chapter 4) may be catalogued for subsequent invocation from a batch stream. How these processes are performed is a matter for control over the use of the system, rather than use of explicit privileged facilities being provided.

NIPS/FFS

The conditional expressions and report specifications used in the interrogation function (see Chapter 4) and the specifications used in the update function (see Chapter 5) may be stored in a library. No explicit privileged facilities are provided for control over these functions.

TDMS

Report descriptions specified using COMPOSE (see Chapter 4) and MAINTAIN task descriptions (see Chapter 5) may be stored by the user for subsequent invocation from a terminal. Central control over what is stored is a matter of control over the use of the system.

UL/1

Computational procedure which are to be used frequently may be stored with the stored data definition in a use of the Establishment division (see Chapter 6) or the Revision division (see Chapter 3). Control over the use of these divisions is a matter of control over the use of the system.

COBOL, DBTG

No centralized capability is provided. However the use of the COPY clause to store source statements in the COBOL library, may be centrally administered.

IDS

Storage of programs is an operating systems function. The data administrator must play an administrative role in controlling this storage.

IMS

The data administrator must select frequently used programs for inclusion in the library of such programs. Their names must agree with a master list of permitted programs. The terminals which are allowed to invoke a program are defined using the security maintenance utility, which generates a security matrix. This matrix determines which programs may be executed from which logical terminals.

SC-1

The program library facilities of the operating system is used to store host language programs and self-contained function. The capability for storing queries and report definition is provided as part of the interrogation function (see Chapter 4). In the update function, all transaction data definition and transaction program definition must be prestored in a library maintained by the update function (see Chapter 5).

8.2.6 Control over scheduling algorithm

Only the data administrator, with his superior knowledge of the whole set of applications,can assign priorities to the various programs to be executed. The number of priority levels may vary and indeed may itself be a parameter controllable by the data administrator within the constraints of the system.

The techniques for controlling priority schedules are inherent in many operating systems (see Chapter 10) and are a part of the data base management system only in so much as it supplements existing and inadequate operating system capabilities. The use of priority scheduling may depend on whether the data base management system is being used in a dedicated mode on the machine or whether it is competing for machine resources with other classes of processing.

GIS, MARK IV, NIPS/FFS, TDMS, UL/1

No capability is provided.

IDS

Capability is provided in the GECOS III operating systems for controlling the priority level of application programs. It is also possible to specify "privity" for a program, namely that it is considered a component of the operating system.

IMS

There are fifteen priority levels provided. Some transaction programs may be specified as batch message processing programs which means that they are to be run in batch mode at some later point in time when the systems administrator initiates the execution of a batch stream.

The priority scheme includes facilities for assigning a limit to the number of items in a queue, and a corresponding limit priority. If the number of transactions in the queue with a given priority is exceeded, then the priority is changed automatically to the limit priority in order to cause it to be run sooner.

COBOL, DBTG, SC-1

No capability is provided.

9. STORAGE STRUCTURE

Storage structure is the user data, as actually stored on physical media. It must be distinguished from the data structure (see Chapter 2), which is the user's conception of the data, in terms of its logical relationships. A single data structure can be stored in different ways, resulting in different storage structures, and in fact, many systems select from several different storage techniques, depending on how the data is to be used. A given basic storage structure may be implemented in various ways, on various secondary storage devices. The nature of the storage structure is further conditioned by the characteristics of those devices, and of the operating system.

The data is ultimately stored as a continuous string of bits or characters, and the system must set up correspondences between logical data elements, such as item and groups, and the portions of the string which constitute their instances. The nature of this correspondence differs from level to level in the structure. The method used to move from one element to another logically related one also varies with the level.

To improve efficiency of accessing the stored data, the user may be given some degree of control over the storage organization, either overall, or on a detailed element-by-element level.

9.1 Techniques in item and group level storage

Given a string of characters representing, say, an assembly (set of group instances), the system must be able to isolate the individual group instances and item values. A system can accomplish this by using one or more techniques analogous to those associated with system languages (see 1.4). These are:

- An element (item or group) appears at a known position in the string.
- The elements are separated by some distinctive separation symbol.
- Each element is accompanied by a symbol which identifies or names it.

The ways in which these are used, and the complexity of the resulting relationship between logical structures and stored data, are determined to a large extent by the variability allowed in various item and group attributes, for instance

- item value length (see 2.1.2.2)
- item value existence (see 2.1.3)
- number of members in an assembly (see 2.2).

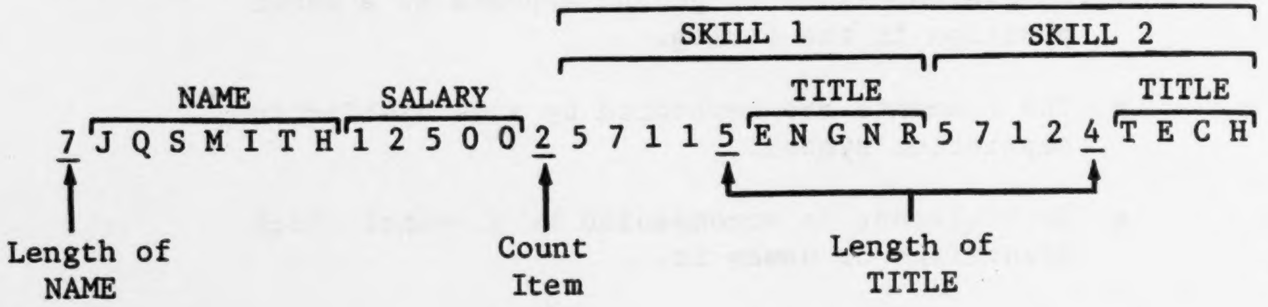
If all attributes are fixed, the system can in effect set up a table giving the correspondence between item name, and position in the string of the first character of the value (position, that is, relative to some known point, such as the beginning of the group).

Such a table might effectively have the form

Item schema name	Starting character position
INITIALS	1
LAST NAME	3
SALARY	25
1ST SKILL	30
. . .	

This approach may require more space, since room has to be left for the longest possible value of each item, and may restrict the number of occurrences of a group to a predefined maximum. In a variation of this method, each entry contains an individual table of item value starting points for that entry.

A method of solving the variable length item problem is to store the length of the value immediately before the value itself in the string. Similarly, the number of group instances in an assembly may be stored as a data value (often called a "count item value") preceding it. Use of these techniques is shown in the following example of an entry:



The system can determine the starting points of items and groups and the length of the entry by using the underlined values, and the lengths of fixed length items. With this technique, a length of zero could be used to indicate a missing or null item value.

Another method is to insert <u>separators</u> or <u>delimiters</u> of different kinds to mark off portions of the string of bytes, for example:

```
(ED)J(SD)Q(SD)SMITH(ID)12500(ID)5711(ID)ENGNR(ID)5712(ID)TECH(AD) ...
    ↑    ↑               ↑      └──────────┬────────┘└─────────┬─────────┘
 Sub-items             Item      Group instance      Group instance
 (first and            Value └────────────────────┬──────────────────────┘
  middle                                        Assembly
 initials)
```

where (ED) = entry delimiter
(SD) = sub-item delimiter
(ID) = item value delimiter
(AD) = assembly delimiter

The system can scan down the stream and find any particular part of the data. Non-existent data is handled simply by leaving it out; for example, if a title had not yet been assigned to skill 5712:

```
... (ID) 5711 (ID) ENGNR (ID) 5712 (ID) (AD) ...
```

If items or groups are often non-existent, it may be advantageous to use a tag technique in which a recognizable specially encoded item or group identification (name, label) is stored preceding each value:

```
(I1)JQSMITH(I2)12500(G1)(I3)5711(I4)ENGNR(G1)(I3)5712(G1)
 ↑          ↑        ↑                                ↑
Item       Item     Group                           Value
code       code     code                         non-existent
                                                   for I4
```

Now the system can scan down the string, identify each portion of the data, and ascertain where data is missing by comparing the tags with the stored data structure. For example, above, an I3 (job code) is normally followed by an I4 (title), but in this instance was followed by a G1 indicating that I4 was not present.

9.1.1 Item level storage

Item values are stored as binary configurations which are binary or binary-coded-decimal for numeric item types; and according to a code, usually ASCII (USA Standard Code for Information Inter-Change) or EBCDIC (Extended Binary-coded Decimal Interchange Code) for string item types.

Packed decimal numbers (two 4-bit digits per 8-bit byte) usually have a sign, if present, in the four bits to the right of the least significant numeric digit. Unpacked numbers (1 digit per byte) typically carry the sign in the left-hand four bits of the right-most byte (usually called zoned decimal).

In putting information in an area in storage larger than necessary to contain it, numeric information is usually placed as far to the right as possible ("right justified") and unused space on the left is filled with zeroes; while string-type data is usually left-justified and right-filled with blanks. These conventions for sign placement and justification will be assumed unless otherwise stated. The information already presented under Data Structure (see 2.1.1, and Figures 2-3 and 2-4) will not be repeated.

GIS

Null values are represented by binary zeroes. Item values are not aligned with 32-bit memory word boundaries.

MARK IV

Item value storage depends on system environment.

NIPS/FFS

The entry and repeating group identifiers and the constituent items of non-repeating groups must be alphanumeric; that is, numeric items which have binary values are illegal in these places.

The length of a variable length item is stored in the first word of the block of numeric items which are not members of non-repeating groups. Coordinate type data is stored as two 32-bit binary words (one is for latitude and one for longitude), each with a 10-bit integer and a 22-bit fraction.

TDMS

Item values are stored in the form of a pair:

item-identifier,item-value

If the value is non-existent, the whole pair is absent from the data. If an item value is numeric and less than 7 digits, it is converted to a 21-bit binary number. Otherwise it is stored in the form of its actual input character string (including decimal point), preceded by the number of characters in the value, and the item-value in the identifier-value pair is a pointer to that representation.

Items of type "date" are stored as a 21-bit number of days since 1900-01-01.

UL/1

String and coded item values can be stored in either fixed or variable length modes.

The determination of storage mode for each item is based upon an analysis of the input data at the time the file is created. If the item values are sufficiently constant in length across the complete set of values in the initial instance of the file, the fixed mode is selected. Otherwise the variable mode is assigned. In the latter case, the length is contained in one or two bytes (depending on the item schema) immediately preceding the value. A value stored in fixed mode is right-filled with spaces.

A "date" type item is stored as century and year (12 bits), month (4 bits), and day (8 bits).

COBOL

Item storage is an implementation function. In a typical system, 2-, 4-, and 8-byte fixed; 4- and 8-byte floating point; and ASCII or EBCDIC string values are provided (see 2.1.1).

The user may have the ability to specify that a code other than the system standard is used in a given file, and to have it automatically translated accordingly.

DBTG

Item storage will be defined by each implementor of the DBTG proposal.

IDS

Item storage is according to COBOL usage.

IMS

IMS treats all stored item values as byte strings.

SC-1

A binary integer value is filled on the left with sign bits. An item of type "general" is preceded by its length, as a one-byte binary integer. A null value for a numeric item is represented by zeroes, and for a string item, by blanks, unless the item has been specified to be optional or variable. In that case, there will be no value for it in the file. Lengths of variable length or optional items are stored in the entry header.

9.1.2 Item storage in groups and entries

Unless otherwise stated, item values in a group or entry instance are stored contiguously and in the order shown in the definition of the constituents of the group or entry schema.

Systems generally do not make any special provision for the storage of non-repeating groups. Such a group is stored as the concatenation of its constituent item values, and is, with its constituent items, accessed in the same way as other items.

GIS

Item values are accessed via the Data Definition Table (DDT) which is referenced by the compiler which translates function requests.

MARK IV

Access to item values is through compiled instructions, based on the File Definition.

NIPS/FFS

The set of values for the principal items in the entry (the "fixed set"), and each instance of each repeating group ("periodic set"), is individually stored in separate physical records, as is each instance of a variable length non-repeating group ("variable set").

Each such record contains the following information, in the order shown:

Record attributes (length and type).

Information to associate this set of item values with the correct entry, and group within that entry

Entry identifier
Group name (see 2.2.3.1)
Group sequencer (see 2.2.3.2)

Number of words of binary data. (This includes the length of the variable item, if present, and user-supplied numeric items that are outside non-repeating groups.)

Padding for word alignment, if necessary.

Number of characters in the variable field, if used.

User-supplied numeric items. (All numeric items not contained in non-repeating groups are placed together here with one item per binary word.)

User-supplied fixed length alphanumeric data. (This includes alphanumeric items and groups, and numeric groups. These data are stored in EBCDIC.)

User-supplied variable length item.

If this is the record for a "variable set" instance, the numeric items and fixed length alphanumeric data are not present.

Accessing information to the above is carried in a File Format Table (see 3.8 and 9.3).

TDMS

The item-identifier/item-value pairs comprising a group are stored together as part of a body of data called CDATA. Access to item values is through a chain of tables whose structure is described in 9.3 which also describes CDATA.

UL/1

All item values subordinate to all the groups in an assembly are stored together (not immediately following their parents, as in other systems). Within such a collection all items from a given item schema are stored, followed by all those from the next, and so on (see 9.3).

COBOL

Item values in a group are not referenced in the data base as such, but only after the group on entry has been moved from secondary to primary storage. Values are then referenced directly via operand addresses.

DBTG

Storage of item values in groups and entries is determined by the implementor.

IDS

Item storage is according to COBOL usage.

IMS

IMS does not deal with individual items within groups, except in their role as group identifiers.

SC-1

Item values are accessed by simultaneous use of tables derived from the data definition, and such elements as the item value lengths stored in the entry header.

9.2 Techniques in entry and file level storage

The logical data structure is formed by associating lower level schemas to form higher level ones (see Chapter 2). In an analogous way, item and group instances are collected to form higher level groups, and the entry.

The overall form of the storage structure at this level arises from four factors:

> The characteristics of the physical storage medium and its device.
>
> The way in which the correspondences among logical data elements are reflected in their storage.
>
> The way in which logical elements (groups and entries) on the one hand are related to physical records on the other.
>
> The operational environment (see Chapter 10).

9.2.1 Media and devices

Physical secondary storage devices may be sequential or direct access. In either case, the data is generally stored in the form of records, where a record is the portion of data which is manipulated as a single unit or entity by the combination of hardware and software. Some data base management systems make a distinction between physical record, which is a hardware-determined element, and logical record, which is a software entity. In a sequential device, there is a one-dimensional medium (usually magnetic tape), and to move from one place in the file to another, it is necessary to traverse all the intervening information. This establishes a natural corresponding order for the records, which appear one after the other in a sequence on the medium. There is generally no sort of individual identification for the record other than that implied by its relative position in the sequence of records; in other words, there is no "record address".

A direct access storage device (for example, disc, drum, or magnetic strip) is one in which it is possible to reach any part of the data in approximately the same time. Each record in the storage device has a hardware address related to its position on the medium, and can be accessed directly, given that address. In addition, the system may give each record an identifier which also uniquely addresses it. In fact, several layers of these may exist, with each in turn being converted into a lower-level one by the data base system, the operating system, or the hardware, until the final one is reached. In the discussion in this chapter, "address" will be used to mean an identifying element at any of these levels. When an address is stored as an element of data in the system, it is often called a pointer.

The records in direct access storage are not arranged physically in sequence in the same way as the records on a sequential medium, but such a sequence can be defined in terms of the records on consecutive sectors, tracks and surfaces on a disc or magnetic strip, or consecutive tracks on a drum; it then takes on the characteristics of a sequential medium.

9.2.2 Storage structure organization

The two primary elements of the relationship between data structure and storage structure are:

- How the storage structure represents the relationships among entries in a file, groups in a group relation, items and assemblies in an entry, and groups in an assembly.

- How this representation is used in accessing information. A given basic structure of the stored data may be utilized in several different ways for different requirements.

Often an entry (in the form of the collection of all its item values) is stored as a single independent entity in the file, and is accessed as a unit, after which its constituents are available for processing. Alternatively, the group (rather than entry) instance may be the independently-stored element. In most cases, the organization of the entries in a file centers around a set of one or more item schemas taken in an order of significance, which constitute the entry identifier or sequencer.

Accessing an element in the file may be done directly, by furnishing an address or system-provided identifier, or by locating it through its relationship to some other element. In the latter case, the relationship may be that of being next in a sequence of such elements, or it may be a more complex one.

These methods apply not only to the organization of elements in a file, but also to lower levels, for instance the storage of a group as the concatenation of its item values. These techniques can also be applied in various combinations, to allow several different accessing methods within the framework of a single storage structure.

9.2.2.1 Direct storage structure organization

A storage structure using direct access to each element (that is, location of the element by having its address immediately available) can be organized in several ways. One way is to store a new element (for example, group or entry) at any vacant place in the file, saving the address of that location for future use in retrieval. Here the address is in fact used directly, without being related to the values of any items in the stored element.

Accessing can, on the other hand, be based on such values, as in randomizing or "hashing", in which a set of one or more item values in the element is transformed into a record address. This transformation is often based on manipulation of the characters in the value(s). Since its random nature may result in different values transforming to the same address, provision has to be made for overflow from that location into other records. A somewhat related method depends on mathematical calculation of a unique address from the given values.

A different technique is the use of various kinds and levels of indexes, in which the system maintains a list relating values of items to addresses of elements containing those values. This is discussed later in more detail (see 9.2.3).

9.2.2.2 Relational storage structure organization

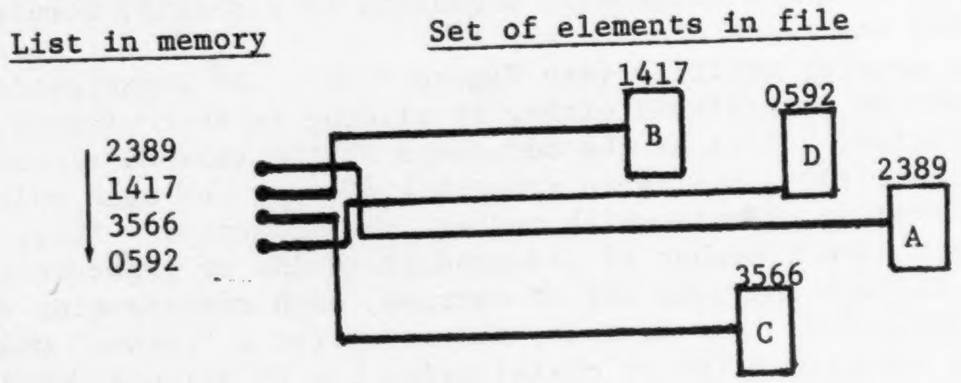
This organization depends on the ability to move from one element, which has been located, to another, related one. The elements in a relation may be associated by being in a sequence, or by provision of a more general structure, in which a number of individual relations exist from one element to others. The ordering of elements in a sequential structure is often according to the value of one or more pieces of data in them, such as a file of entries corresponding to employees, in sequence by employee number.

A sequential structure may be of two forms:

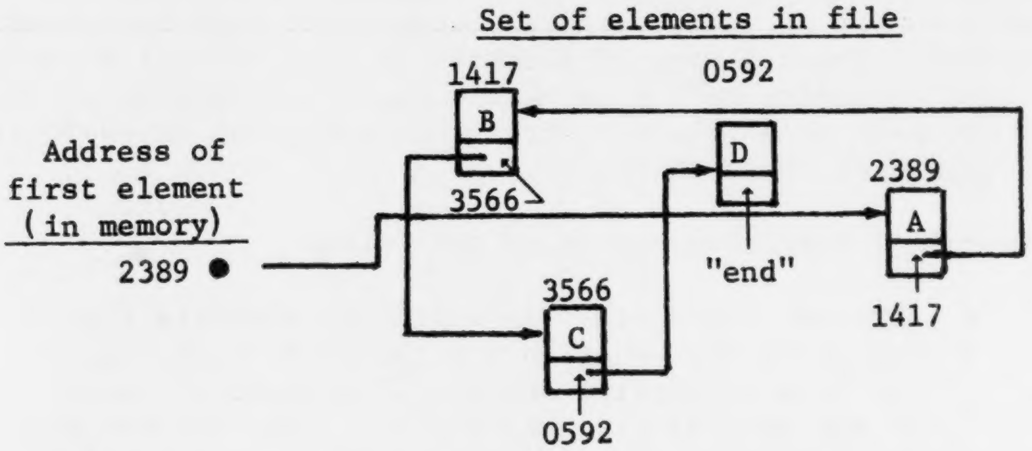
- Physical sequential, in which the elements follow one after the other in a sequential store, which may be a sequential device, like magnetic tape, or may be a direct access device used sequentially. An example is writing the groups in an assembly on tape.
- Logical sequential, in which the elements are separated in the store (generally direct access), and the sequence is established through other means.

In any sequential organization, the system must be able to tell when it has reached the end of a sequence of related elements (for example, the groups in an assembly). This may be accomplished by storing the number of elements in the set (the "count item" for an assembly), or by placing a special "end" symbol in the last element.

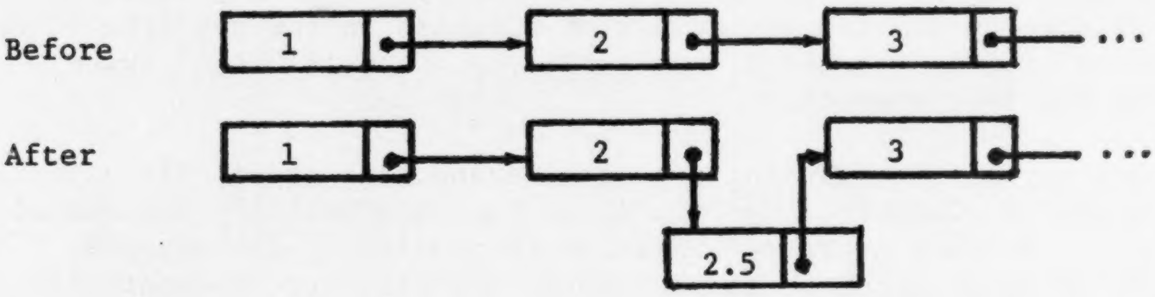
One way of establishing a logical sequential organization for a set of elements in a file is to have a physically sequential list of their addresses immediately available, for example in internal memory. To go through the structure sequentially, the system moves down the list of addresses, using each in turn to access the next element:



Another way to implement a logical sequence is to set it up as a chained organization, in which there is a way of locating the first element of a collection, and each element contains the address of a pointer to (that is, address of) the next.



Modifications to an element in a file which is on a physically sequential store may require rewriting all or part of the whole file. The logical sequential organization avoids this. If a record needs to be added, it can be placed in any empty location in the store, and the pointer list or chain modified accordingly:



Instead of having a collection organized in a single, purely sequential way, each element may have several different relations to other sets of entities (see Figure 2-32). An organization of this kind can be achieved either by placing in that element, for each relation, a list of the addresses of the related elements, or by establishing one chain from that element for each relation, or by providing indexes which reflect the structure. Thus, a file might have a number of independent chains or separate lists running through the same set of entries, each representing different relations. For example, the entry for a "person" might have one relation (list or chain) tying him to all his "projects", another, to his "skills", and a third to his "children".

This kind of facility can also be used to order a single file in different ways, by having one chain connect the entries in employee number order, and another, alphabetically by name.

A file schema of this sort consists of a number of different groups, linked together in a complex way (see Figure 2-31). Since, so to speak, any part of the file is accessible from any other, processing of related information is accomplished by moving through the file from one group to another, following the relations. If the process is to begin with a specific occurrence, then a direct access method must be used, or part of the collection must be searched. In other cases, it may be satisfactory to retrieve all the instances of a given schema, in an arbitrary order.

9.2.3 Indexes

The use of indexes was mentioned earlier as a means of associating a data value with the address of an element containing that value. An index may consist of the set of value-address pairs, or, more compactly, of a value and the set of addresses associated with it. In either case, the index is usually in sequence by value. If the index is large, it may be broken up into several levels, with the first one, in high-speed memory, giving for each range of values the address in the file where a finer index to that range is located; that one in turn may lead to another even finer one, and so on.

An index may contain one value-address pair for every occurrence of a given value, or it may represent only unique values, or even only ranges of values. In that case, the address is a pointer to a single element with that value, or in that range, and the rest are located through a relational structure.

If the entries in a file are organized only sequentially by entry identifier, finding the entry with a given identifier would require searching the file from the beginning. This can be avoided by providing an index which gives the file addresses for certain identifiers occurring at intervals along the sequence. An entry with a particular identifier is located by searching the index for the largest value which is still less than that identifier, accessing the corresponding entry, and then following the sequential organization (which may be physical or logical) until the desired entry is found. This is termed an indexed sequential organization.

Even though entries in a file are easily accessible through their identifiers (via direct or indexed sequential methods), it may be desirable to locate an entry according to the values of some item other than the one determining the basic file organization. In that case, to avoid the necessity of searching all or most of the file, the system may set up secondary indexes for values of other items. A file which is indexed by every value of every item is sometimes called an inverted file. An inverted file can be applied to find entries which satisfy a combination of conditions. For example, if all "employee" entries with SKILL=5711 and DEPT=2013 were needed, the address lists for those two values in the index could be compared. Addresses appearing on both are those of entries satisfying the condition.

9.2.4 Relation of records to logical elements

Data is stored in the form of records, whose characteristics are determined by the system hardware and software. The data structure, on the other hand, is reflected in entries, assemblies, groups, and other logical elements.

The correspondence between records and logical elements varies. In the simplest, each entry or each group of the storage structure corresponds to exactly one record, and vice versa (see Figure 9-1a). Other possibilities are to store multiple or partial entries per record (see Figure 9-1b,c).

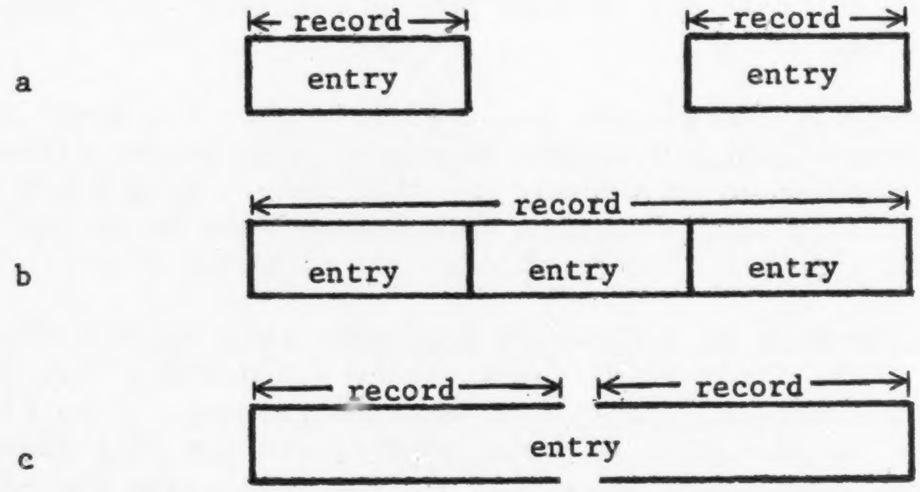


Figure 9-1
Relation of records and entries

It should be noted that systems may use one of these techniques for groups, and a different one for entries. Also they may be used in various combinations. Further, entries and records can be fixed or variable length.

Some systems assign a fixed set of elements to a record, but allow them to be rearranged within the record to meet changing requirements. In many systems (usually at the operating system level), an element may be assigned to a record initially, but may be subsequently moved to another record, with the original one preserving a link to the new location.

Addition of a new entry to the file is accomplished either by storing it directly (see 9.1.2.1), or putting it in a vacant place (unused record or piece thereof) in the file, and adding the address of that place to any appropriate lists, chains, tables, etc.

Pointers in a storage structure are generally addresses of records, but they may be used to relate both

- records, for example, to chain them together to accommodate a sequentially organized file of entries, and
- entries or groups, for generalized relational structures (see 9.2.2.2).

In the latter case, if a record contains more than one entry or group, the pointer is used to locate the record, which is then searched for the desired element.

9.3 Entry and file level structure

Similar techniques can be used to relate the groups in an assembly, assemblies in an entry, and entries in a file. These have been discussed in the preceding section, and the current section describes which are used in the surveyed systems.

A problem arises in mapping a hierarchical structure of data (that is, a tree) into a sequentially organized storage. The most common technique is to follow each element by the first element at the next lower level if there is such an element, and by the next element at the same level if there is no lower level one. This results in a top-down organization.

Since storage of non-repeating groups is generally similar to item storage, only repeating groups will be discussed here, unless otherwise specified.

GIS

Each group from a given schema has a constant length; an assembly consists of the concatenation of the groups forming it, in order by the values of the user-defined group sequencer (see 2.2.3.2).

Each group which has one or more dependent group assemblies contains a user-defined count item for each such assembly, whose value is the number of members in the assembly. The entry consists of the assemblies, in top-down order.

Each entry is stored as a logical record (padded out to fill if necessary) in the OS/360 data management sense. Physical records may be fixed or variable length, and may contain one or more logical records (but always an integral number), as determined by the user (see 9.5).

The organization of entries in the file is also chosen by the user, and may be sequential or indexed sequential [using OS/360 access methods SAM (Sequential Access Method) and ISAM (Indexed Sequential Access Method) respectively] with the identifier item(s) of the root group (the entry defining group) serving as the sequencing element. Storage is on tape or disc, with ISAM being usable only on the latter, and requiring fixed length records. Variable length logical records are permitted with SAM.

MARK IV

Groups in an entry are stored in a top-down fashion, with an integral number per physical record. If a repeating group schema is defined to have a fixed assembly size, its superior group does not have a count item. Fixed and variable size assemblies may be nested in any order.

A variety of accessing methods are used, depending on user requirements.

NIPS/FFS

The entry consists of a sequentially organized set of physical records, all with the same basic internal structure. The content of each data record is either the set of values of the principal items in the entry, an instance of a repeating group in the entry, or an instance of a variable length non-repeating group. The initial portion of each data record contains a value defining its length, a code (to distinguish it from various kinds of non-data records which are also part of the file), and the value of the identifier for the entry to which it belongs.

The record further contains the number of the group schema of which it is a part and the value of the group sequencer for this instance. If the record contains the principal items, the group schema number is zero. If no group sequencer is specified by the user, the system supplies an internal one.

The records in the file are in sequence by record type, entry identifier, group schema number, and group sequencer, in that order, so that the file consists of entries in identifier order; the entries are made up of assemblies in group schema order; and the assemblies are a contiguous set of group instance physical records in sequencer order.

The file contains at the beginning a non-data format description record for the overall structure of the entry, with the length and other attributes of the set of principal items, and of each repeating group. A format record also exists for each item and non-repeating group schema, with accessing and other information derived from the data definition.

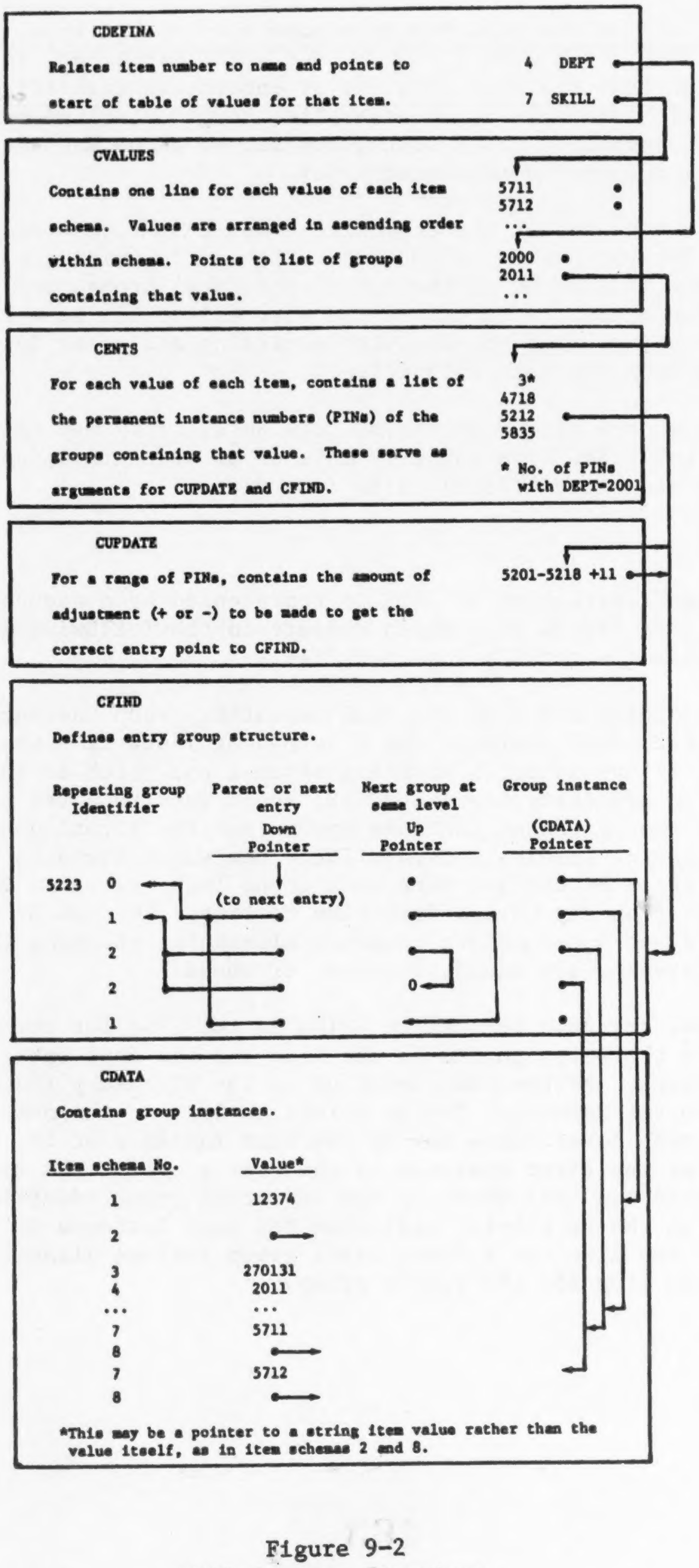
Data files are stored as OS/360 Data Sets. They may be used as sequential files from magnetic tape or as sequential (SAM) or indexed sequential (ISAM) files from disc.

TDMS

Entry level structure in TDMS is represented by a sequence of tables (see Figure 9-2) which operate in the following way, and which define a totally inverted file.

CFIND contains one line for each repeating group instance in the file. Each such instance has a permanent instance number which is used in the system's indexing scheme, and which is the argument for an intermediate table CUPDATE, which furnishes the offset between the permanent instance number and the location of the corresponding information in CFIND. The value found in CFIND is the location of the instance in a group instance table CDATA; this location may change from time to time. The use of CUPDATE and CFIND as intermediate elements eliminates changing the PINs when instances are added, deleted, or moved.

As shown, the down pointer in CFIND in the line for the principal items in the entry points to the line for the next entry; at lower levels, to the next level up in the hierarchy ("down" here means to the parent). The up pointer points to the next instance at the same level (this may be the next instance of the same group, or the first instance of the next group at the same level; the system can tell which by the repeating group identifier). A zero in the up pointer indicates the last instance at a given level. The line for a lower level group follows directly in CFIND the line for the parent group.



CDEFINA

Relates item number to name and points to start of table of values for that item.

4 DEPT
7 SKILL

CVALUES

Contains one line for each value of each item schema. Values are arranged in ascending order within schema. Points to list of groups containing that value.

5711
5712
...
2000
2011
...

CENTS

For each value of each item, contains a list of the permanent instance numbers (PINs) of the groups containing that value. These serve as arguments for CUPDATE and CFIND.

3*
4718
5212
5835
* No. of PINs with DEPT=2001

CUPDATE

For a range of PINs, contains the amount of adjustment (+ or -) to be made to get the correct entry point to CFIND.

5201-5218 +11

CFIND

Defines entry group structure.

Repeating group Identifier
Parent or next entry
Down Pointer
Next group at same level
Up Pointer
Group instance
(CDATA) Pointer
5223 0
1
2
2
1
(to next entry)
0

CDATA

Contains group instances.

Item schema No.	Value*
1	12374
2	• →
3	270131
4	2011
...	...
7	5711
8	• →
7	5712
8	• →

*This may be a pointer to a string item value rather than the value itself, as in item schemas 2 and 8.

Figure 9-2
TDMS Storage Structure

CDATA (and all other tables that make up the data base) are stored as contiguous blocks (records) of fixed length (2160 bytes) on random access storage. Directories within the file contain the starting block number for each table type so that the various pointers can be converted into an absolute file block number for direct single access retrieval.

UL/1

The storage organization is according to these rules:

> All items at one level in an entry are stored preceding all items at the next level.
>
> All item values for one multiple-valued item schema are stored together.
>
> All fixed length items are stored preceding all variable length ones.
>
> All items subordinate to any group in an assembly are stored together, preceded by a set of count items, one for each member of the higher level assembly.

The effect of these rules is best illustrated by an example. The data structure is:

A				Entry
	B			Item (variable length - V)
	C			Item (fixed length - F)
	D			Repeating group
		F		Item (F)
		G		Item (F)
		H		Item (V)
	E			Repeating group
		I		Item (V)
		J		Repeating group
			K	Item (V)
			L	Item (F)

An instance of the entry might be

```
B
C
  F  G  H    Group )
  F  G  H    Group } assembly
  I
     KL
     KL
  I
     KL
     KL
     KL
  I
     KL
```

The stored data for that instance would be arranged as follows (the data is stored as a continuous string; the gaps are for clarity):

$$\underbrace{\mathrm{CvB}}_{1}\quad\underbrace{2}_{2}\quad\underbrace{\mathrm{FFGGvHH}}_{3}\quad\underbrace{3}_{4}\quad\underbrace{\mathrm{vIII}}_{5}\quad\underbrace{231}_{6}\quad\underbrace{\mathrm{LLLLLLvKKKKKK}}_{7}$$

Explanation

1 Root group. The fixed length item C is followed by an index table, v, for the variable length item, and then by the variable length item B itself.

2 This specifies that the values following represent a two-member assembly of group D (items F, G, and H).

3 Level 1, group 1. All values from the fixed items F and G precede all those for the variable item H.

4 This is another count value (similar to the preceding "2"), which defines the next assembly of group E (single item I) to have three members.

5 Level 1, group 2. There is no fixed length item in this group.

6 Each of these three count values is associated with the corresponding one of the three values of I, and indicates how many of the following L's and K's belong to that I.

7 Level 2, group 1.

Each variable length item consists of one or two bytes containing the length, followed by the value. The index table, v, for variable length items, precedes the set of variable length item values in the assembly. It contains two bytes for each item in the set of values which follows it; those two bytes give the relative position of the start of that item (that is, of the length byte(s).

The entry, as stored, consists of system data (date stamp information, for instance); a hierarchical directory; and the data itself, stored as described above. The hierarchical directory gives for each repeating group schema the relative starting position in the stored data of its fixed and variable areas.

A physical record consists of an integral number of entries, each of which starts with a four-byte value which indicates the length of the entry.

COBOL

The storage structure for COBOL data is dependent upon the implementation. The following describes a typical case.

Groups in an assembly are stored contiguously, with the size of the assembly being either fixed, or specified by the value of a count item in the parent group or entry. (The name of this count item is given in the data definition in the "DEPENDING ON" item name phrase of the OCCURS clause.) Storage of assemblies is top-down.

Each OCCURS clause specifies a maximum number of members for an assembly; the system allows space for this maximum amount of data, except for the last group schema in the entry, when only as much space as is needed is alloted. The result of this technique is that each group begins at a fixed relative position in the data stream, eliminating the requirement for excessive computation of positions, or searching down the data. Each entry is contained in a separate physical record. Organization of a file is sequential or indexed sequential, based on the entry identifier; or direct access, with the user having responsibility for determining the address in the file where an entry is to be stored, and for saving the addresses for future use in retrieval of the entry.

DBTG

No storage structure implementation is specified, but certain elements can be inferred.

Each entry is stored as a separate entity, with a unique system-provided identifier called the data base key. The user may specify the accessing method for the entry schema as

- DIRECT, in which for each access a specified location is loaded with the data base key (provided by the system when the entry was first stored in the file).
- CALC, in which an identifier is calculated from data values in the entry.
- VIA, in which the entry is accessed as a member of a dependent assembly in a particular group relation.

In the first two cases, the entry is accessed directly, using the identifier as an "address", in the general sense described previously. In the case of the group relation, the assembly may be defined to be organized in either of two ways:

- CHAINED, in which the parent group contains a pointer to the first group in the assembly, each group of which in turn has a pointer to the next, except the last group, which points back to the parent. If the OWNER option is specified, every group, not just the last one, points to the parent. If the PRIOR option is exercised, each group also points to the preceding one, making the chain traceable in either direction (with the PRIOR pointer from the parent leading to the last group).

- POINTER ARRAY, in which the parent group in the relation contains or has associated with it a list of the addresses of (pointers to) all the groups in the dependent assembly. The PRIOR function can be achieved by moving to the previous pointer in the array. The OWNER function is moot, since the array is associated with the owner instance. The pointers in the parent group may point to other pointer arrays, which point to still others, and so on, until the data is reached.

A variation is the

- DYNAMIC POINTER ARRAY, which differs in that the instances in the relation may be of any group schema in the system, not just the one(s) declared as dependent; the array is destroyed at the end of the run unit which created it, and there are no pointers from the dependent instances back to the parent.

The dynamic pointer array allows the host language programmer to manipulate a temporary collection of data (such as transactions) without regard for the defined data structure.

Although only a single group relation has been discussed, each group schema may take part in any number of group relations as parent or dependent, and therefore a group may have any number of chain pointers or pointer arrays. Also pointers may be associated with, rather than contained in, groups.

The relationship of logical elements to physical records is an implementation function.

IDS

The storage structures at the file level is based on group relations with assemblies organized as chains. A group may be stored directly, with a system-provided identifier, or with one based on randomization of data values. The storage is divided into pages, whose size is defined by the user at file creation time, generally to be an integral number of contiguous physical records.

Each page contains an integral number of groups (generally with some space left over). Each group has associated with it a line number, and one of the sections of the page header is a table which relates line numbers to starting character positions of the groups on the page. The combination of the page number and line number is the unique group identifier used in chain pointers. If one group needs to be expanded, other groups on that page may be moved around to make available enough contiguous space for it; values in the line number table then have to be changed to reflect the new positions of the groups on the page.

When the direct access method is used, a number of group identifiers may randomize to the same page number. A page header contains a pointer to the start of a chain which runs through all groups which randomize to that page. The groups themselves are stored within the page if space is available, or in a nearby page.

IMS

Each stored file is called a physical data base. The storage structure used for a physical data base depends on the accessing method to be employed (which is specified by the user), and will be described in that context. A pointer in IMS may be either the address of a physical record in the file (similar to other systems); or an item value which can be used to locate a record, such as the identifier of an entry which has been stored using a direct method (see 9.2.2.1). Also in the Direct Access methods, the pointer not only selects the record, but specifies a particular starting character position.

Four access methods are available in the system:

- HSAM (Hierarchical Sequential Access Method)
- HISAM (Hierarchical Indexed Sequential Access Method)
- HDAM (Hierarchical Direct Access Method)
- HIDAM (Hierarchical Indexed Direct Access Method)

The storage structure characteristics associated with each of them will be described in turn.

- HSAM

  This method is used with either sequential or direct access devices. The storage organization is physical sequential, by entry identifier. Groups are concatenated in storage to form an assembly, and assemblies in turn, in a top-down fashion, to form an entry. Each group contains a code to indicate its schema, and groups are stored in the same order as that in which their schema definitions were input.

  Each physical record contains an integral number of groups; it is fixed length and the unused area is filled out with zeroes. This organization uses the OS/360 access methods BSAM (Basic SAM) and QSAM (Queued SAM).

- HISAM

  This method is used with direct access devices. The storage organization is indexed sequential, by entry identifier, but otherwise follows the HSAM methods.

  An additional feature is provided by the secondary data set group. The user may choose a group schema at the level just below the root group, and have its instances set up in an indexed sequential storage structure of its own, separate from the root group structure. This secondary structure is organized and indexed by the concatenation of the root group and second level group identifiers. This technique allows a second level group to be accessed directly without going through the root group instance and other dependents.

  OS/360 ISAM (Indexed Sequential Access Method) is used, supplemented by OSAM (Overflow Sequential Access Method).

- HDAM

  This method is used with direct access devices and makes use of OSAM. The root group of each entry is stored according to a user-supplied randomizing method (see 9.2.2.1) applied to the entry identifier. The root group, and as many dependent groups as possible, are stored in the directly accessed record; the remaining groups in the entry are placed in records (in an overflow area) which are chained to the root-group one. Groups are stored in order of input rather than in a top-down hierarchical manner (made possible by the internally stored schema identification). If more entries randomize to a record than can be contained therein, they also are stored in records chained to the initial one. In order to reduce the length of such a chain, it may be broken up into a number of shorter ones, all originating in the initial record. In that case, entry identifiers are randomized to a record address and "anchor-point number", where the anchor-point number designates which one of these shorter chains receives the entry. In this case, each directly addressed record has one chain for each such anchor point, resulting in several short chains instead of one long one.

Secondary data set groups can be established for groups at any level, not just second-level. Groups in an entry are chained together; the user specifies whether this is to be done in a top-down way (dependent groups follow the parent; the last group in one assembly is chained to the first group of the next higher level assembly) or in a combination of top-down and left-to-right (a group is chained both to the first of its dependent groups and to the next group at the same level). The latter gives faster access in traversing the data, at the expense of space for the additional pointers required.

- HIDAM

  This method is used with direct access devices. The root groups are accessed via an indexed sequential technique rather than directly. Other aspects are the same as for HDAM.

SC-1

Storage of assemblies is in top-down form. The system maintains tables which record the size of assemblies for accessing purposes.

Each schema in the system has associated with it an ordinal number representing its relative position in the set of elements dependent on the next higher element in the hierarchy. Thus the data base has number 1, the fifth file in the data base would have number 5, and so on. A repeating group assembly and its schema or prototype instance ("record") are considered to be separate elements at adjacent levels. Each schema in the system is also given an item class code (ICC) which is the ordered set of schema numbers starting at the data base and proceeding down the hierarchical path until that schema is reached. (An 'R' is used at each point a prototype instance ("record") of a repeating group is encountered. The 'R' stands for an instance number within the assembly.)

Thus in Figure 2-11, if "PERSON" represented the entry in the personnel file, which was the fifth file in the data base, the ICC for TITLE would be 1.5.R.4.R.2 (that is, data base 1, file 5, group or item 4, item 2). If SKILL were a non-repeating group, the ICC of TITLE would be 1.5.R.4.2.

Also, each instance in the data has an IPC, which is formed by appending to the ICC a set of instance number values, one for each R-value in the ICC. Thus in Figure 2-12, if this were the 496th record in the file, "SALESMAN" would have the IPC 1.5.R.4.R.2.496.3 since "SALESMAN" is associated with the 3rd instance of "SKILL".

Physical records are fixed length (within one file), say two to four thousand bytes, chosen by the user. Item instances thus may cross record boundaries. Each physical record is associated with the IPC of the first element it contains, and its starting character position. The records are grouped into volumes in the OS/360 sense, and the volumes into data groups, both as specified by the user. The collection of data groups forms the file. Introduction of the intermediate level of the data group allows copying on only that portion, rather than the whole file, when father-son updating is done.

The system maintains a directory for the IPC values in each data group and volume, and each volume has a tree-structured directory which relates IPCs to physical records.

9.4 Data base level storage structures

Among the surveyed systems, GIS, MARK IV, NIPS/FFS, UL/1, COBOL, IMS, and SC-1 allow multiple file schemas to be considered as a data base. Only in IMS is this association reflected in the storage structure.

IMS

The storage structures described above (see 9.3) related primarily to the "physical data bases" set up by the data administrator. A user can also establish a "logical data base" consisting of sub-sets and concatenations of the groups from the physical data bases (see 2.2.2.1, 2.4.2.1, and 3.10). When the storage structure of the physical data bases are defined, they must have provision for all of the group relations needed to implement the logical structures. Most of these provisions take the form of pointer fields which set up logical connections between groups within the same or different physical data bases. The result of these linkages is a complexly organized network of physical data bases forming a system data base; each user, however, sees his logical data base as no more complicated than a tree.

9.5 User control of storage structure

In many systems, the user is given some degree of control over the storage structure at the file level, for instance, by being able to specify whether the file is organized according to a sequential or direct access method. The user may also be allowed to indicate that different elements of data are likely to be required at the same time, so that the system can arrange for them to be "close" in the file. Another element of control is over which items are to be the subject of secondary indexes, and how those indexes are to be arranged.

GIS

Following the definition of the root group in the file definition, a statement of the form

$$\text{DATM: DSORG=}\left\{\begin{matrix}\text{PS[U]}\\ \text{IS[U]}\end{matrix}\right\}\text{, [keyword=parameter] ...}$$

allows the user (e.g. data administrator) to specify to OS/360 Job and Data Management whether the file is to be organized in a physical sequential (PS) or indexed sequential (IS) fashion; and also that it contains data whose addresses are used as pointers, and which are therefore unmovable (PSU or ISU). The keyword-parameter pairs following permit the user to state (among other things):

- Average and maximum physical record length in bytes.
- Entry length (fixed; or maximum if variable length)
- Recording device, mode and density
- Amount of space on the file medium to be assigned to this file, and size of the increment to that basic allotment
- Whether the index for indexed sequential is to be put in the main file area or kept separate
- Whether physical records are fixed or variable length
- Whether a physical record contains one, or more than one entry.

MARK IV

The user may specify the file organization (sequential or indexed sequential, with fixed or variable length records), and record size and blocking. He may also specify that the file is to be organized under any of the access methods supported by IBM's DL/I.

NIPS/FFS

Storage structure control is handled by statements in OS/360 Job Control language.

TDMS, UL/1

None

COBOL

The statement

$$\text{SYNCHRONIZED}\begin{Bmatrix}\text{LEFT}\\ \text{RIGHT}\end{Bmatrix}$$

allows the user to specify positioning of an item value within an area in memory.

FILE-LIMIT IS address-1 THRU address-2

allows the user to designate a portion of a direct access device to contain the file.

Storage organization is specified by

$$\text{ACCESS MODE IS}\begin{Bmatrix}\text{SEQUENTIAL}\\ \text{RANDOM}\end{Bmatrix}$$

If RANDOM, then the user must also specify

ACTUAL KEY IS address-location

DBTG

The DIRECT, CALC, and VIA options in the LOCATION MODE statement specify how the entries will be accessed (see 9.3). Further, however, VIA also specifies that the entry will be placed as cloe as possible to the logical insert point in the named group relaton.

In the definition of a group relation, the statement

SEARCH KEY IS item-name-1 [,item name-2] ...

will result in the establishment of an index, by the concatenation of the item names, to all groups in each dependent assembly in the relation. Organization of the assemblies in a group relation is specified by

$$\text{MODE IS} \left\{ \begin{array}{l} \text{CHAIN [LINKED TO PRIOR]} \\ \text{POINTER ARRAY [DYNAMIC]} \end{array} \right\}$$

IDS

The user may define the IDS page size (in characters) and total number of pages in the IDS file by

MD file-name; PAGE CONTAINS number-1 CHARACTERS:

FILE CONTAINS number-2 PAGES

If the user knows that a particular group relation is likely to be used often, a statement

PLACE NEAR group-relation-name CHAIN

may be included in a group schema definition. The system will then try to optimize the storage of the dependent group in the relation, for example by placing the assembly on the same page as the parent group, which allows retrieval of the assembly without additional file accesses.

A statement

PAGE-RANGE IS number-1 TO number-2

can be used to limit storage of groups of a given schema to a specified region within the total file. When the user wishes to update the file, the operating system will allocate only the specified region for updating. This subfile definition capability allows concurrent update of the total file in a multiprogramming environment.

The user may distribute groups of a given schema evenly over the entire file by saying

INTERVAL IS number PAGES

which normally operates only during file creation, and causes the specified number of pages to be skipped between storage of successive groups.

IMS

Storage structure control at the file level is defined by

```
     DBD  NAME = file-name, ACCESS = method

          [,RMNAME  = (randomizing-procedure-name

               [,no.-of-anchor-points] [other-data])]
```

where "access" can be LOGICAL (specifying that a logical data base is being defined) or the name of any of the four access methods available with the system. "Other-data" limits the part of the file to be randomized to, and governs the placement of data into file areas.

DATASET statements in the definition control assignment of groups to secondary data set groups, and record length and blocking factor in the file.

A set of statements and options are provided to control the establishment of the storage structure necessary to support the formation of logical data bases from physical ones.

SC-1

User control of storage structure is done by a job called Data Location Definition which allows the user to add additional volumes to a data group, or to create a new file, and specify how the file is to be divided into data groups, and the data groups into volumes. He may also create a new generation of only one data group in the file. Another element of control is whether the index to the file is written out with it. The user may specify the number of bytes in a physical record by a BLOCKSIZE statement, and the percentage of empty space to be left when the file is created by a SLACK phrase.

In defining a "file" (that is, repeating group assembly) at any level, the user may specify the increment to be used in assigning "r-values" to successive instances.

The user may, in connection with the definition of an item schema, specify

$$\text{INDEX} \quad \left\{ \begin{array}{ll} \text{NONE} & \\ \text{ALL} & \\ \text{LIST} & \text{value-list} \\ \text{RANGE} & \text{range-statement} \end{array} \right\}$$

to cause production of an index of some or all values of that item.

10. OPERATIONAL ENVIRONMENT

The operational limitations imposed by hardware and external software not included as part of the data base management system are the principal parts of the operational environment. The factors that together describe this environment are highly interrelated, and require both an analysis of the systems in their environment, and the related operating system, communication, and other software.

Versions of the COBOL compiler have been implemented on many different hardware systems. As a result, no single description of one of its environmental interfaces would be representative. The DBTG proposal has not yet been physically implemented. Thus neither of these systems is included in the discussion in this chapter.

10.1 Hardware environment

Data base management systems may be designed to function on only one computer system, or a family of computer systems, or even on completely different computer systems. The hardware aspects of the operating environment include restrictions, limitations, and flexibility as they affect the user. Aspects of the environment under discussion include the minimum central processor configuration on which the data base management system can operate, internal core, direct access storage devices, sequential storage units required by the data base management system itself, and any limitations on the number of terminal devices and direct access storage devices that can be accommodated by the central processor and its associated operating system. Limitations which these impose on the size of the user's data base are also considered part of the hardware environment. Many systems can operate with additional hardware; i.e. more than that required in a minimum system. The additional supported hardware is also an important part of the environment.

10.1.1 Processor, main memory, and special requirements

The minimum hardware configuration, both central processor and core, which is utilized to support the systems is shown in Figure 10-1. Sometimes it is found that the minimum configuration merely allows space for the operating system of the particular machine plus the particular data base management system without provision for buffers.

Thus the minimum configuration is normally given for operation of only one function for one user against a single data file, with no provision for concurrency of operations. The minimum memory for on-line versions is generally greater than that for batch because of the additional routines for special input/output devices, and sometimes the additional buffers for multiple users.

SYSTEM	PROCESSORS	MINIMUM MEMORY		REQUIRED HARDWARE OPTIONS
		BATCH	ON-LINE	
GIS[1]	IBM 360/40	128k bytes	256k bytes	decimal arithmetic
MARK IV[2]	IBM 360/25	48k bytes	n.a.	decimal arithmetic
NIPS/FFS	IBM 360/40	128k bytes	256k bytes	decimal arithmetic
TDMS	IBM 360/50	256k bytes	256k bytes	real time clock, drum storage, fetch protect feature.
UL/1	Spectra 70	256k bytes	n.a.	none
IDS	H 615	64k words	64k words	none
IMS	IBM 360/40 or 370/145	256k bytes	n.a.	none
	IBM 360/50 or 370/145	n.a.	512k bytes	none
SC-1	IBM 360/50	256k bytes	n.a.	none

[1] Recommended configuration for MFT is model 40 with 196k bytes, for MVT is model 50 with 512k bytes.

[2] Subset systems of MARK IV require only 32k bytes.

Figure 10-1
Minimum CPU and core requirements

10.1.2 Data base storage media

Different systems require or support different types of magnetic tape or direct access (e.g., disk, drum) devices for the storage of the user's data base(s). In the systems studied, data bases are stored on magnetic tape or on direct access device according to some file level storage structure (see Chapter 9). Provisions may be made for incorporating additional devices beyond those required by the minimum hardware configuration.

In order to allow backup (for restart or recovery from data base damage) it is often necessary to store a log of transactions (see 8.2.3). The storage needs for this, such as extra tape drives, are also given in Figure 10-2.

Often part of the available storage is needed by the operating environment for its libraries. This may leave little or no room for storing data. The second column of Figure 10-2 therefore deals with the storage devices required to support the operating system alone, while the third column shows the additional requirements for the data base management system. The final column shows other secondary storage supported for data base storage.

10.1.3 Terminal equipment

Various types of terminal equipment are used with on-line systems, and often a user can interface with the data management system from some remote location by means of a communication link, or a small computer can be used as a remote input/output device.

The terminal equipment that can be used by the various systems is shown in Figure 10-3. The table excludes any terminal used as an operator's console. The number of terminals that can be active at one time usually depends on the type of communication link, and the type and number of multiplexers selected to support the system. The location and number of terminals are shown in the last three columns of the table.

10.2 Software environment

The software environment of a data base management system involves both the operating system which is necessary for operation of the system, and other basic software using or used by the data base management system, such as procedural language compilers, report writers, and sort routines.

The interface between the data base management system and the operating system is normally explicitly defined, because a change in the environment of the system need not unnecessarily invalidate previous procedures or data. Normally many facilities of the operating system are used by the data base management system, but some are specifically excluded.

SYSTEM	REQUIRED STORAGE DEVICES		SUPPORTED MEDIA FOR DATA BASE
	OPERATING SYSTEM ONLY	ADDITION FOR MINIMUM DATA BASE SYSTEM	
GIS	1 IBM 2314 disk volume	3 IBM 2311 disk drives	any device supported by OS/360 sequential and indexed sequential access methods
MARK IV		1 IBM 2311 disk	any device supported by OS/360 sequential and indexed sequential access methods
NIPS/FFS	varies with hardware configuration and choices made at system generation time.	3 IBM 2311 disk	magnetic tape or disk; data files (including "File Format Table") and updating transaction programs may be stored as sequential files on either tape or disk; files to be queried or updated from a terminal must be disk files.
TDMS	1 IBM 2311 or 2314 disk drive 1 tape drive	1 IBM 2302 or 2 IBM 2311 or 2 IBM 2314 disk drives	random access devices only
UL/1	1 RCA 70/564 disk drive 3 tape drives	1 RCA 70/564 or 590 disk drive	RCA 70/564 and 590 disk drive
IDS	1 DSU 270 disk drive	none	any device supported by the GECOS III operating system
IMS	1 IBM 2314 disk volume	1 IBM 2311 disk drive 1 IBM 2400 tape drive	any device supported by OS/360; the tape drive is needed for logging transactions, etc.
SC-1	1 IBM 2314 disk volume	1 IBM 2314 or 2 IBM 2311 disk drives	any device supported by OS/360 sequential and direct access methods

Figure 10-2
Data base storage media

SYSTEM	TYPE OF TERMINAL	NUMBER ACTIVE	LOCAL	REMOTE
GIS	any terminal supported by the "queued terminal access method" (QTAM) or "telecommunication access method" (TCAM); this includes IBM 2741 typewriter and IBM 2260 display	depends on the "message control program", which specifies terminal network	yes	yes
MARK IV	none	n.a.	n.a.	n.a.
NIPS/FFS	IBM 2260 display	unlimited, except through terminal processor storage requirements	yes	yes
	IBM 2250 graphic		yes	no
	IBM 2265 display		no	yes
	IBM 2741 typewriter		no	yes
	IBM 1050 typewriter		no	yes
TDMS	IBM 1052 typewriter	10 (ten) total, limited by operating system	yes	yes
	IBM 2741 typewriter		yes	yes
	IBM 2260 display		yes	yes
	Teletype model 33 and 35		yes	yes
	CCI/30 display		yes	yes
UL/1	none	n.a.	n.a.	n.a.
IDS	typewriter devices: Teletype models 33 35, and 37 Friden 7100 IBM 2741 typewriter processor devices: G115 processor UNIVAC 1004 G225/235 processor	for one H 355, either 192 typewriter devices, or 16 wide band processors, or 32 processors may be connected; up to four H355 devices may be used	yes	yes
IMS	IBM 2740 typewriter	the total number cannot exceed 255	no	yes
	IBM 2740 model 2 typewriter		no	yes
	IBM 1030 typewriter		no	yes
	IBM 1050 all components		no	yes
	IBM 2260 display		yes	yes
	IBM 2780 card reader/ printer		no	yes
SC-1	none	n.a.	n.a.	n.a.

Figure 10-3
Terminal equipment

10.2.1 Operating environment

One of the principal effects of the operating environment is found in its method of scheduling a group of programs or run units. Any single program may be in one of four states:

- Running; i.e., being serviced by a CPU;
- Blocked; i.e., unable to run because it is waiting for some other action, such as a data base read, to be completed before it can be rescheduled;
- Waiting; i.e., ready to run, but in a queue until CPU or other resources are available to it; it will enter into execution when it has the highest priority of all other waiting jobs. A special case of waiting is "entering", where the program has not previously been in the running state.

The CPU or CPUs will therefore take a program and execute it for a period of time which will then result in one of the following:

- the execution of the program is completed;
- the program is in a blocked state;
- the amount of time ("quantum") that is allocated to the job is exhausted, and it returns to the waiting state.

Depending on the design of the operating system, there may therefore be one or many programs co-resident in the machine, and these may run concurrently or separately. They may also run for a fixed quantum of time and be terminated for rescheduling, or be rescheduled only when blocked.

The following working definitions are used:

A uniprogramming system is one in which a program is initiated, and the scheduler component of the operating system does not start another program until the first one is completed. In this case, the waiting state is never reentered once the program has started.

A multiprogramming system allows all four states of a program to exist. It can be implemented with different features:

- Dismissal of a program when it is completed, blocked, or has exhausted its time quantum;
- With more than one program physically in the main memory, or using "roll in/roll out" techniques.

A multiprocessing system is one where there is more than one processor. Each processor may be working on different (or the same) programs simultaneously. Each CPU of the multiprocessing system can adopt the characteristics of either a uniprogramming or multiprogramming system.

The elapsed time of running a program in a data base management system (i.e., the total time between initiating a program and receiving a response, or completion) will depend on the operating environment and the total number of users.

Those significant and relevant features of particular operating systems used by the data base management systems are shown in Figure 10-4.

OPERATING SYSTEM	SIGNIFICANT FEATURES
ADEPT - 50	multiprogramming 10 simultaneous jobs (4 tasks each), time-sliced, roll in/roll out
DOS	multiprogramming, 3 co-resident jobs
GECOS III	multiprocessing, roll in/roll out, 16 co-resident tasks, time-sliced
OS/360: MFT II	multiprogramming, 15 co-resident tasks
MVT	multiprogramming, 16 co-resident tasks, roll in/roll out
PCP	uniprogramming
TDOS	multiprogramming, 6 co-resident tasks

Figure 10-4
Operating system characteristics

There are two entirely different ways in which the integrity of the data base may be threatened:

- By two application programs within the data base management system both attempting to change the data base at essentially the same time;
- By some other program, external to the system, attempting to change the data base.

Protection against the first of these is normally provided by the data base management system (see 7.5.2.4 and 10.2.1.2). Protection against the second is normally vested in the operating system, and its fulfillment is shown in Figure 10-5 under the heading "data base integrity."

If the operating system has a feature such as multiprogramming, the data base management system may make use of this feature. This can mean that the implementation of an additional feature in the operating system affects the total handling of data. As an example, handling interrupts and scheduling programs is generally a function of such an operating system. A message processing facility in the data base management system may supplement this by subsequently scheduling its own programs. The use of the operating system and/or special scheduling within the data base management system is shown in the last column of Figure 10-5.

SYSTEM	OPERATING SYSTEMS	DATA BASE INTEGRITY	SCHEDULING
GIS	OS/360 (MFT II, MVT)	uses OS/360 to stop non system access	uses OS/360
MARK IV	DOS or OS/360	uses OS/360 to stop non system access	uses OS/360
NIPS/FFS	OS/360 (PCP, MFT II, MVT)	uses OS/360 to stop non system access	uses OS/360 augmented for terminals
TDMS	ADEPT-50	multiple users may use, except for UPDATE, where only one use is permitted	uses ADEPT-50 multi queue time sharing
UL/1	TDOS	uses TDOS	uses TDOS
IDS	GECOS III	uses GECOS III to protect on concurrent access to a common file	uses GECOS III
IMS	OS/360 (PCP, MFT II, MVT)	uses OS/360 to stop non system access, also has 'HOLD' feature	uses OS/360, but IMS transactions are scheduled by IMS scheduler
SC-1	OS/360 (PCP, MFT II, MVT)	lockout is at the file level; if the file is over several reels/volumes, it is at the volume level	uses OS/360

Figure 10-5
Operating system environment

10.2.2 Concurrency of operations

There are three distinct functions that can be performed by a system user: create, interrogate, and update. In a multiple user environment, where multiprogramming or multiprocessing techniques are used, two or more users may be running programs which endeavor to perform one or more of these functions concurrently on the same or different portions of the data base. If the system is only one of several programs running on the hardware, then the conflict may be extended to several different systems trying to access the same data base concurrently.

If concurrent running of the same program is to be allowed, the program is normally implemented in re-entrant code. Concurrent operations on the data base may occur because:

- the data base management system allows more than one user to call simultaneously on the same or different functions;
- the operating system allows more than one user to interact with the same copy of the data base management system;
- the operating system allows more than one copy of the data base management system.

The need for several programs to share resources such as core or secondary storage data base can cause conflicts which end in deadlock; thus if two programs both need more core before completing and cannot give up any core until completed, they both wait for the other to terminate, and consequently neither completes. A more common deadlock in data base management systems occurs when two or more users are waiting for each other to release portions of the data base. The possibility that two co-resident programs conflict or communicate with one another again allows contention (e.g., where several users are concurrently modifying the same part of a data base), which requires a guarantee that the integrity of the data base is not violated; such conflicts therefore impose restrictions on the scheduling and security of certain programs.

10.2.2.1 Concurrency during file creation

Of the three functions create, interrogate, and update, the creation process must be completed prior to another operation on the same file, whereas interrogation and update may be concurrent with themselves or one another. The only questions on concurrency of creation are therefore:

- whether two files can be created simultaneously;
- whether the creation process on one file can be achieved at the same time as other functions on another file.

The answers to these two questions are given in Figure 10-6.

SYSTEM	SIMULTANEOUS CREATION	CREATION WITH ANOTHER FUNCTION
GIS	by using OS/360	by using OS/360
MARK IV	by using OS/360	by using OS/360
NIPS/FFS	yes, by updating null files from terminals; otherwise in batch using OS/360	yes
TDMS	yes	yes
UL/1	no	no
IDS	by using GECOS III	yes
IMS	by using OS/360	by using OS/360
SC-1	by using OS/360	by using OS/360

Figure 10-6
Concurrency of the create function

10.2.2.2 Concurrency with single copy

When only one copy of the data base management system exists, there are several ways that concurrent interrogation and/or update may occur:

- One application program called by two users, who interact with different data files;
- One application program, interacting with the same data file, called concurrently by two users;
- More then one application program interacting with different data files;
- More than one application program acting on the same data files.

Some systems have means to achieve these, others either cannot operate in this fashion, or do not protect files if the condition occurs.

Updating is often an exclusive operation so that no other activity can take place while updating proceeds. A system, however, may initiate functions such as remapping storage that are carried on while updating is performed. In other cases, user initiated actions such as interrogation or reporting can take place on the same file while updating is in progress.

GIS

Only one program or user may apply any single function to a single data file at any one time. The LIST statement may be used to print reports during update runs.

MARK IV

File access is limited only by operating system rules. Any user can, therefore, access a file being updated, even for independent updating. A single user may combine "transaction" updating, "Processing and Record Selection" logic, and report writing using the "Output Content" form.

NIPS

In batch processing one function can be performed on a file by one application program at a time. When terminals are used, interrogations and updating for the same or other files can take place concurrently, except when the temporary file produced by updating (which contains new entries or repeating groups or those with changed data values) is being merged into the file.

Much of this concurrency is obtained because, even when using prestored interrogations or updates, each user operates with his own copy of the interrogation or update.

TDMS

Not applicable, since normal operation of the system involves multiple copies.

UL/1

Only one programmer or user may apply a single function to a single data file at any one time.

IDS

This system is not written in re-entrant code, and therefore concurrent running of a single program is impossible. The system supports only one data base (for each copy of the system), but a data base may be split into subfiles. It is not possible to concurrently interrogate and update, but on a subfile basis it is possible to apply concurrently one function (for each subfile). Thus if there are n subfiles, it is possible to provide either n interrogates, or n updates concurrently, or any combination thereof.

IMS

Any one application program can access or update only one file. Simultaneous interrogation and update is allowed both for different files, and also for the same file except when the HOLD feature (see 7.5.2.4) is used.

SC-1

Only different programs may access different data files concurrently.

10.2.2.3 Concurrency with multiple copies

When more than one copy of the data base management system is allowed within the operating system, a further dimension of concurrency is possible. The ability to limit data file access to one particular copy of the system is usually vested in the operating system.

GIS

Multiple operations of any kind on different files are available through OS/360 multitasking.

MARK IV

Multiple operations of any kind on different files are available through OS/360 multitasking. By having shared direct access storage, it is possible for several users to interrogate the same file.

NIPS/FFS

Multiple operations of any kind, using shared data bases, are possible using OS/360 (PCP).

TDMS

The normal operation of this system involves multiple copies of the program. Multiple concurrent interrogation is allowed on the same file, but there is no concurrency of update; i.e. one update is run on its own for any one data file. For multiple files, concurrency of updates and interrogations is possible.

UL/1

No concurrency is possible on the same file, but both interrogation and/or update can be performed concurrently on different data files.

IDS

It is possible to have concurrent updates and access on both the same and different data files, except the restriction that there be only one update per subfile still applies.

IMS

Multiple operations are possible through use of OS/360.

SC-1

Normal operation involves multiple copies of the program. Although concurrent interrogation is allowed on the same file, there is no concurrency of updates of one file. It is possible to have concurrent update and access on both the same and different files, except that only one update is possible for any one subfile.

10.2.3 Modes of system use

The data base management system may use, or operate, within its environment in either batch or on-line mode. In the batch mode, the user submits his job, with all necessary control statements and/or inputs, and receives an answer or solution based on the inputs and program involved. In an on-line mode the user may submit all his data at the same time as he requests that the program be executed, but he may also be allowed to give some data later, based on interim results, or as an answer to a question generated in the program.

On-line use may be restricted to the local area in the immediate vicinity of the CPU or may be in a remote area, connected to a CPU from some distance, often over leased or dial-up phone lines.

All systems can operate in a batch mode, which implies that the jobs or application programs that are to be run by the system are held in some queue until resources become available, and they are scheduled. Two different types of batch mode operations can be distinguished. Jobs are:

- queued in a first-in first-out mode;
- sorted into some new order before running.

All systems studied have the former capability but none perform any special sorting prior to execution.

On-line use means here that the execution of a program or job is terminal oriented, and that single requests are executed (though the operation may be multiprogramed or multiprocessed by the system). The Local column in Figure 10-7 refers to the use of local terminals other than the operator's console, for job submission.

SYSTEM	ON LINE	
	LOCAL	REMOTE
GIS	yes	yes
MARK IV	no	no
NIPS/FFS	yes	yes
TDMS	yes	yes
UL/1	no	no
IDS	yes	yes
IMS	yes	yes
SC-1	no	no

Figure 10-7
Modes of system use

10.2.4 Software facility interfaces

10.2.4.1 Operating system

Data base management system design is normally affected by the previously specified operating system. Input/output control subsystems may be a part of the operating system, or else can be developed primarily for the data base management system. Because the data base management system may run under different operating systems, the method of accommodating various operating systems affects the user.

A further aspect of the interaciton between the operating system and the data base management system is in the method of identifying and using the physical files. Functions usually performed within the operating system, such as the provision of indexes or directories, will affect the way that the data base management system is implemented or performs. Similarly, the process of opening and closing files is often performed by the operating system rather than the data base management system.

The basic input/output facilities of the operating system (often referred to as access methods) are normally used by the data base management systems; however, the access methods are often augmented to provide better indexing for hierarchical or other more complicated files. Space and resource management are normally controlled by operating system functions. This includes allocation of buffers, and opening and closing files. Figure 10-8 shows other operating systems characteristics which may or may not be included.

SYSTEM	ACCESS METHODS	TASK SCHEDULER	LOADER LINK EDITOR	SPACE AND RESOURCE MANAGEMENT	COMMUNICATION FACILITIES
GIS	yes	yes	yes	yes	yes
MARK IV	yes	no	no	yes	n.a.
NIPS/FFS	yes	yes	yes	yes	augmented
TDMS	yes	yes	yes	yes	yes
UL/1	yes	no	no	yes[1]	n.a.
IDS	yes	yes	yes	yes	yes
IMS	augmented	yes	yes	yes	augmented
SC/1	augmented	yes	yes	yes[1]	n.a.

[1] Provides own buffering but uses O.S. open and close.

Figure 10-8
Use of operating system facilities

Additional features:

GIS

None

MARK IV

Report files which are sorted need the OS SORT/MERGE package in a separate job step from interrogation. See also 10.2.4.3.

NIPS/FFS

Processing of source language statements requires the F level Assembler and the MACLIB library that contains macros used to produce generated code.

Generated programs are loaded using the services of the Linkage Editor and require the use of the JOBMACRO library for the execution of macros that are part of the generated code. The OS/360 Sort program is also used for the execution of RASP sorts.

User supplied programs are of two types, those added to the Terminal Processor at system generation time and those supplied as subroutines to be executed under the control of the system. At the time the Terminal Processor is generated so it will fit the installation's particular terminal configuration, user problem programs, conversational programs, and graphic programs for use from the terminals may be added. Subroutines have a standard calling sequence so they may be written in any 360 programming language but they should be constructed as a single root segment, contain no I/O, and be available from a user's partitioned data set library.

TDMS

The system is written in JOVIAL, and thus a JOVIAL compiler is required. An interface is available, through JOVIAL, for programs to be constructed that use the retrieval routines.

UL/1, IDS, IMS, SC-1

All use the operating system facilities for compiling programs for user predefined procedures.

10.2.4.2 Communications subsystem

The interface between user and the system may involve remote processing. The design of this interface affects the user as he makes contact with the data base management system, enters his program with the data, and signs off. The interplay between the operating system and the data base management system in transaction management can significantly affect the way in which a program is executed. In some systems, the polling and queueing of messages is done by the operating system, and in others the major scheduling of users is in the hands of the data base management system, and priorities are given in an entirely different manner.

GIS

The processing from remote terminals is controlled by user written message control programs of the OS/360 queued telecommunicaitons access method (QTAM) or telecommunications access method (TCAM). The terminal user interface and the queueing and priority of input are all determined by the design of these message control programs.

MARK IV

None.

NIPS/FFS

Terminal input and output operations are controlled by a Terminal Processor Monitor. This monitor uses the OS/360 Graphic Access Method (GAM) or the Basic Telecommunications Access Method (BTAM) for actual transmissions to and from the terminals. The monitor maintains disk input and output message queues for each terminal. Input queues are scanned for the program requested, and if the program requested is acceptable, the request is passed on to a Terminal Processor Supervisor.

The output queues can be displayed at graphic terminals by a program that can serve all terminals concurrently. This part of the monitor displays a screen of data under the direction of user requests. The data in view may be moved backward or forware one screen load or one line at a time. These move requests are serviced by the monitor without entering them into queues or referring them to the Terminal Processor Supervisor. The monitor can also call on a utility program to write the contents of a message queue on the system output file, any specified printer, or any stored data file.

The Terminal Processor Supervisor acts as the interface between programs and the terminal input and output. The supervisor has two modes of operation. At system generation time the supervisor's mode of operation must be specified for each program that can be requested by a terminal user. The terminal interrogation program, QUIP, illustrates one mode of the supervisor's operation. When it is used in conjunction with an MVT operating system or MFT operating system with the subtasking option, it is executed by being ATTACHed to the supervisor as a subtask. All terminal requests concurrently use that part of the supervisor that reads input message queues a line at a time, and writes output message queues and terminal messages. When the last QUIP output to a terminal has been written in the output message queue, the QUIP subtask is detached from the supervisor.

If QUIP is used with an MFT operating system that does not have the subtasking option, QUIP is linked to the Terminal Processor Supervisor and requests are processed serially. Concurrent processing of requests can be achieved in such a system by using a separate copy of the Terminal Processor Supervisor in each core partition.

The other mode of operation for the Terminal Processor Supervisor is illustrated by the terminal update program SODA. The first request for SODA causes the creation of a supervisor subtask. Subsequent requests are chained onto this task in a last in first out manner. Regardless of the number of terminals operating on a file, a single "hold file" of updated records is used. When a terminal user indicates the file is to be updated from its "hold file," the terminal is "signed off." When all terminals have been signed off from SODA its subtask is detached from the supervisor.

TDMS

ADEPT-50 permits operation of any filed program by a user specifying the program name. All TDMS operations are filed by name as programs within ADEPT-50. All terminal communications are handled by ADEPT-50 which manages the input/output for TDMS. The language of TDMS is essentially free form with most specifications having a command form. Each command begins with a command word (on a few occasions implied) and is followed by command objects and modifiers where applicable. Most commands are designed so that the order in which they are entered is not important.

Although provisions have been made in some operations to accept prestored transaction statements, most of the input is expected to be via interactive console.

The carriage return symbol is used as the terminator of a command. The asterisk is available when the user wants to continue a command beyond one line. The question mark is available to the user at any time to find out what kind of information can be legally used at the spot where it was input.

Each command is error checked at the time it is entered and any errors det ected at that time are immediately displayed to the user for correction before the command is accepted.

UL/1

None.

IDS

This is programmed by the user.

IMS

IMS uses prestored transactions. The format of input is: a transaction code, optional password, and optional data.

Three types of processing are allowed: "Message" is for messages received from remote terminals. "Batch-message" is batch oriented processing with references to on-line files. Batch-message processing may indirectly communicate with remote terminals. "Batch" processing is strictly batch oriented and may communicate with neither on-line files nor remote terminals.

All message and batch-message programs are defined to IMS and identified as to type at IMS system generation time. The programs are identified to the system by PSBNAME (program specification block) and have an associated transaction code or codes. The transaction codes are the means by which the user identifies, to IMS, the program he wishes to execute. The PSBNAME and hence the transaction code are tied to the processing programs through a PSB generation. The PSB generation also describes the files the program may access, and the type of processing that may be done against those files.

All messages received from remote terminals are stored in an IMS input message queue. Transaction codes associated with message processing programs cause the associated program to be scheduled for execution under the control of the IMS nucleus as soon as adequate core is available. To this end a fixed amount of message processing core is allocated to the IMS system when the IMS system is started. The scheduling of a job in IMS depends on three factors:

- its priority number;
- the "limit number" of other jobs in the queue which have the same category of priority;
- the "limiting priority".

If the job has the highest priority, it is run next, except when the latest addition exceeds the limiting number of jobs at the given priority. If the addition of the new job will exceed the limit, then the priority is automatically switched to the limiting priority (which normally is larger) and hence the job will be scheduled earlier (possibly next).

Messages with transaction codes associated with a batch-message program do not cause the corresponding program to be executed but remain in the queue until the operator submits the program as a normal job in the batch stream. The program then interrogates the IMS message queue for the messages which contain its input.

Messages from processing programs, whether message or batch-message are placed in an output message queue. When the line is available the message is then sent to the terminal.

Batch programs are scheduled as batch jobs in the normal stream. These programs have the full range of data base capabilities, but can access neither the input nor the output message queues and run totally independently of the IMS nucleus.

"Command messages" allow the user to communicate with other users and display statistical parameters. The master terminal operator must enable the terminal. The user may "hold" a terminal until his output is returned.

SC-1

None.

10.2.4.3 Other software

Software facilities other than those normally thought of as part of the operating system, may be used by the data base managment system. Typically, the sort/merge package may be used by the extraction facilities of a self contained system to publish a sorted list (see 4.6.4). Sorting may be achieved directly within the host language capabilities (see 7.5.3.4). Normally, the compiler used in host language systems, and for own-code routines in self contained systems, is also part of the operating environment.

SYSTEM	SORT/MERGE	COMPILER
GIS	yes	no
MARK IV	yes	yes
NIPS/FFS	yes, for batch interrogation. no otherwise	yes
TDMS	no	yes
UL/1	no	no
IDS	yes, except for sorting "chains"	yes
IMS	yes	yes
SC-1	yes	yes

Figure 10-9
Use of other software facilities

10.2.5 Procedure preparation, modification, and submission

Within a data base management system, features are normally provided to aid in application program or procedure preparation. Such features include the ability to:

- formulate procedures, requests, or programs for future processing;
- modify such procedures by a text editing capability;
- catalog such procedures within the system for future recall, modification, or submission.

This set of facilities applies to both batch and on-line environments, and may be available for the range of users from application programmers to the parametric user. The level of interaction of these facilities may vary from minimum commands to a tutorial capability.

GIS

Utility task specifications are used to store, maintain, and retrieve users' input line sets, which consist of user statements of up to 120 characters per line, with continuation to other lines. Any set of input lines may be stored by using the format:

```
SAVE  input-line-set-name
```

followed by:

```
END
```

The saved input line set may subsequently be invoked by:

```
CALL  input-line-set-name
```

The input line set may be listed by the specification:

```
LIST TS  input-line-set-name
```

and it may be edited by the utility task specification:

```
CORRECT  input-line-set-name
AT line-number  DELETE  n  ADD m
(m new lines)
AT line-number ...
END
```

The effect of an AT statement is to replace n lines starting at the specified line number, with the m new lines.

The name of the input line set may be changed by:

```
CHANGE   old-name TO  new-name
```

and an input line set may be deleted by:

```
DELETE  input-line-set-name
```

A utility task is also provided for saving users' procedural task specifications in compiled and executable form. This utility is invoked with the statement:

```
SAVEX  saved-procedure-name
```

followed by the procedure to be saved. The saved procedure may be subsequently invoked with the statement:

```
RUN    saved-procedure-name
```

MARK IV

Data submission, including cards describing interrogation logic and report specifications, is governed by OS or DOS/360 Job Control Language conventions.

NIPS

Update-programs may be prefixed to the data files to which they apply. All other cataloging of procedures (JCL statements) and programs are handled by OS/360.

TDMS

The COMPOSE operation includes facilities to construct report descriptions, and the MAINTAIN operation includes similar facilities to build maintenance task descriptions. Both of these contain a set of commands for editing (REVIEW, CHANGE and DELETE) and commands which make it possible for the TDMS programs to have the ADEPT-50 operating system catalog and retain the descriptions is separate files (SAVE and RETRIEVE).

UL/1

The preparation of programs is a purely batch operation, with no text editing capability. The cataloging is entirely within the system, and only computational procedures may be stored (see 4.5).

IDS

Cataloging, text editing, and on-line formulation aids are possible within the operating system.

IMS, SC-1

All procedure cataloging is achieved by OS/360 JCL commands. There is no capability provided, within the system itself, to aid in the formulation and editing of the programs.

10.3 Modes of the operation by data base management functions

The mode of operation describes the various operational methods available to the user for accomplishing his data processing task. It may constrain the user in the way that data and procedures can be entered and information retrieved. Possible modes of operation are: batch, interactive, and transaction.

10.3.1 Batch mode

Batch mode implies that the programs are entered as a contiguous collection, normally called a batch, and that the entire batch will be processed without user or operator interaction. Often these batched programs have some commonality; e.g., they may input different sets of data for a common process. This, however, is not required, and the program may be either prestored, or compiled and then run as part of the batch stream. Batch mode may include the use of presorted programs. Batch mode usually implies batch output and may include controls for validation of a group of programs.

10.3.2 Interactive mode

The interactive modes usually imply processing in the order of arrival or according to a priority scheme based on arrival time. Within the interactive mode there are several recognizably different kinds of interaction. Within each of these, the kind of data and the number of files used depends on system implementation. Also the use of these interactive modes for data management system functions such as data definition, creation, interrogation or updating is a matter of system implementation. A system may supply a type of interactive processing for some or all such functions.

The first type of interactive processing will be called "conversational" mode. It is found in systems that lead the user through the steps of a terminal session, or upon request tell the user what alternatives he has at a specific point in a terminal session. Conversational mode may be provided to specify a particular application (i.e. write a program) for a system function, or for executing one that was prestored in a manner that gives the system access to it.

The second type of interactive processing will be called "prestored" mode. It occurs when the terminal user is allowed to examine data and prestored procedures and to specify execution procedures, data and/or parameters that differ from those that were prestored. Prestored mode differs from conversational mode in that the system does not actively assist the user. The user must know what he can do and how to do it. Prestored mode may be used to specify procedures or to execute prestored procedures, with or without execution time overrides of parts of the prestored procedures.

10.3.3 Transaction mode

The interactive modes presuppose that the user will have some control over the way that a program is executed. Transaction mode implies that a particular transaction identifier with its input data causes the execution of one or more predefined transaction programs. These programs were predefined within the language(s) supported by the system. Normally, each transaction is processed separately, with no reference to the previous or next transaction. Several files can be referenced by the same or different transactions. Transaction mode often implies automatic or semiautomatic data entry, with exception reporting and local validation.

SYSTEM	SPECIFY			EXECUTE PRESTORED		
	CREATE	INTERROGATE	UPDATE	CREATE	INTERROGATE	UPDATE
GIS	B	B or P	B or P	B or P	B or P	B or P
MARK IV	B	B	B	B	B	B
NIPS/FFS	B	B or P	B	B	B or T	B or T
TDMS	C	C or P	C or P	B or C	C or P	B or C or P
UL/1	B	B	B	no	B	B
IDS	B or C or P	B or C or P or T	B or C or P or T	B or C or P	B or C or P or T	B or C or P or T
IMS	B or C	B or C	B or C	no	B or C or P or T	B or C or P or T
SC-1	B	B	B	B or T	B or T	B or T

Table Notation: B = Batch
C = Conversational
P = Prestored
T = Transaction

Figure 10-10
Functions permitted in various modes.

GIS

Task specifications are of three types: data description, procedural, and utility. Data description tasks are used to define the data structure of the user's data base and the input files.

Procedural tasks are used to specify the functions of file creation, maintenance, and interrogation. Task types include: QUERY tasks in which data from up to 16 files may be extracted for presentation in printed reports or for recording in temporary files; MODIFY tasks in which one or more "master" files may be updated from one or more "source" files; UPDATE tasks in which a single master file is updated from a single source file, and CREATE tasks in which a new file is created from a single source file.

A utility task is provided for building and modifying the system security table, which controls access to files and items within files.

The RUN subprocedure may be used to invoke both saved GIS subprocedures and arbitrary user programs. Data may be passed to and from the user's programs through one or more hold files, provided such programs observe the storage structure for these files.

MARK IV

Definitions and report extraction/format specifications may be saved and invoked through OS-DOS/360 Job Control Language and system Run Control and Cataloged Request cards.

TDMS

All time consuming transactions are normally processed in background batch.

UL/1

One UL/1 master file or a coordinated set of up to four COBOL files is used for sequential searches in batch mode.

IMS

There is no limitation on the class of data processing function in any mode.

The definition of data structure to the program is performed as an assembly run in either the COBOL data division, or as a PL/1 statement. The definition of the data structure is an off-line operation utilizing the O.S. assembler and linkage editor. Creation of the file requires allocation of space (using JCL) and populating it later. There is no need for data base definition generation before this allocation. Thus there is independent file creation, data base definition and population.

SC-1

A job definition language is used to create all JCL statements for the user's program to head the data definition and input cards, as well as maintain the job queue file directory with subsumed records that list the steps of a specific user's job, and contains copies of all bindlists for the job.

10.4 System transferability

There are four aspects of transferability that apply to data base management systems. They deal with the extent to which:

- the system can be transferred between computers of the same family operating under the same and different versions of the operating system, or even an entirely different operating system;
- the system can be transferred to operate on computers that are not in the same family;
- the user must modify his procedures when transferring to different computers of the same or different computer families;
- the user must transfer his data base from one type of storage device to another, either on the same or different computers or translate his data from one representation to another.

GIS

The system may be used with any version of)S/360 above the previously stated minimums.

MARK IV

Within the IBM 360 family, the system can operate on the Model 25 and up. JCL changes are, of course, necessary between OS and DOS. DOS/360 does not support variable length ISAM records.

NIPS/FFS

It was initially hoped that the system could achieve hardware independence by being coded in COBOL. The present proportion of COBOL code is so low that the system is operational only with any version of OS/360 on System 360 configurations from Model 40 up.

TDMS

Although the design was intended to be transferable to any hardware system capable of supporting time-sharing, the initial implementation emphasized efficiency for 32 bit word machine.

During design, consideration was given to the possibility of running under various operating systems. To facilitate such transfer, all control operations dealing with program overlays, and all I/O operations have been centralized within TDMS. A major share of the effort in transferring to another operating system would be in rewriting these procedures without affecting the bulk of the JOVIAL code.

UL/1

This operates on models 45, 55, and 60 of the Spectra 70, and on the RCA 2 and 6. It will also operate on the model 46 of the spectra 70 and on the RCA 3 and 7, but not in time-sharing mode.

IDS

Different systems are operational on GE 130, 200, 400 and 600 hardware. They are written in assembly language.

IMS

This system is transferable upward through the 360 and 370 line; it is written in assembler language.

SC-1

The system is designed and implemented to be hardware independent, but the prototype is coded for the IBM 360/50 computer and up, i.e. the functional design is hardware independent. The present version is operating on an IBM 360/65; future plans call for implementations for other computers and application environments.

The same version of SC-1 operates under MFT-II and MVT, but a slightly different version is required for PCP.

APPENDIX - INDEX

Access code, 3.2
  " lock, 2.1
  " (of storage structure), 9.2.2
  " statements, 7.5.2.2, 7.5.2.3
Add (data), 7.5.3.1
Addresses (in storage media), 9.2.1
Administration functions, I.8.8, 8.0
Allocation (of space), 6.3
Assembly (data structure), 2.2
Attribute,entry, 2.4.3
  " ,file, 2.5.3
  " ,group, 2.2.3
  " ,group relation, 2.3.3
  " ,item instance, 4.7.2
  " ,item schema, 2.1.2, 4.7.1
  " ,item value class, 4.7.3
  " ,schema, 2.0
  " ,value class, 2.1.2.2
Audit trail, 8.2.4
Auxiliary data definition, 3.10, 7.6.3
Batch mode (of operation), 10.3.1
Bibliography, 1.8
Binding,compile time, I.5.4, 3.8
  " ,execution time, I.5.4, 3.8
Chained storage structure, 9.2.2.2
Change (data), 7.5.3.2
Close (portion of data base), 7.5.1.2
Command, see statement
Communications subsystem, 10.2.4.2
Composition,data base, 2.6.1
  " ,entry, 2.4.2
  " ,file, 2.5.2
  " ,group, 2.2.2
  " ,group relation, 2.3.2
Compound conditions, 4.3.2
  " group, 2.2
Concurrency of operations, 10.2.2
Conditional expressions, 4.3, 7.5.1.3
Control statements (for programming user), 7.5.1
Conversational mode (of operation), 10.3.2

Counting,cross entry, 4.10.2
Count item, 3.3
Creation, I.8.6, 6.0
  "   action cycle, 6.1
Data administration functions, I.8.8, 8.0
  " base, 2.6
  "   " composition, 2.6.1
  "   " instance, 2.6.1
  "   " schema, 2.6.1
  "   " schema definition, 3.7
  " definition, I.8.3, 3.0
  "   "   ,auxiliary, 3.10, 7.6.3
  "   "   context, 3.1
  "   "   facilities, 6.0
  "   "   (for creation), 6.2
  "   "   form, 3.1
  "   "   interrogation, 4.12
  "   "   processing, 3.8
  "   "   revision, 3.9
  "   "   schema, 3.0
  "   "   storage, 3.8
  "   "   structure, 3.1
  " extraction, 4.6
  "   "   (at the entry level), 4.9
  "   "   (at the file level), 4.10
  "   "   (at the group level), 4.8
  "   "   (at the item level), 4.7
  " independence, I.5.4
  " integrity, 7.4.5
  " manipulation language statements, 7.5
  " mapping, 5.3.2
  " modification statements, 7.5.3
  " retrieval, 5.3.4
  " retrieval statements, 7.5.2
  " security, 3.2, 7.4.5, 8.2.1
  " selection criteria, 4.0, 5.3.4, 7.4.4
  " structure, I.6.2, I.7.1, I.8.2, a.0
  "   "   class, 1.2, 2.0, 2.7
  "   "   definition, I.8.3, 3.0
Decoded value, 4.7.3
Delete command, 7.5.3.3
Delimiters, 9.1
Dependent group, 2.3
Derived data, 4.7.4, 4.8.1, 4.9.1, 4.10.1, 6.5.2
Direct access storage devices, 9.2.1
Direct storage structure, 9.2.2.1
Editing, 3.2, 4.11.1
  "   transaction, 5.3.7
Element, 2.0
Entity, 2.1, 2.4, 2.5

Entry, 2.4
  " attribute, 2.4.3
  " composition, 2.4.2
  " -defining group, 2.4, 3.5
  " ,group, 2.4
  " identifier, 3.5
  " instance, 2.4.2.2
  " name, 2.4.3.1
  " ,plex, 2.4
  " schema, 2.4.2.1
  " " definition, 3.5
  " storage, 9.2, 9.3
  " ,tree, 2.4
  " type, 2.4.1
Error handling (by programming user), 7.4.3
Existence conditions, 4.3.1.7
Extraction, 4.6
  " ,entry, 4.9
  " ,file, 4.10
  " ,group, 4.8
  " ,item, 4.7
File, 2.5, 5.0
  " attribute, 2.5.3
  " composition, 2.5.2
  " data, 5.0
  " definition (for creation), 6.2
  " " (for update), 5.0
  " instance, 2.5.2.2
  " ,inverted, 9.2.3
  " ,linked, 2.5
  " ,mechanized, 4.13
  " name, 2.5.3.1
  " schema, 2.5.2.1
  " " definition, 3.6
  " storage, 9.2, 9.3
  " " ,maintenance of, 5.4.2
  " structure, 2.5
  " type, 2.5.1
  " ,unlinked, 2.5
Group, 2.2
  " attribute, 2.2.3
  " composition, 2.2.2
  " ,compound, 2.2
  " ,dependent, 2.3
  " entry, 2.4
Group, entry-defining, 2.4, 3.5
  " identifier, 2.2
  " instance, 2.2.2.2
  " name, 2.2.3.1
  " ,non-repeating, 2.2
  " ,parent, 2.3

Group, principal, 2.2
" relation, 2.3
" " attribute, 2.3.3
" " composition, 2.3.2
" " instance, 2.3.2.2
" " name, 2.3.3.1
" " schema, 2.3.2.1
" " definition, 3.4
" " type, 2.3.1
" ,repeating, 2.2
" schema, 2.2.2.1
" " definition, 3.3
" ,simple, 2.2
" storage, 9.2, 9.3
" type, 2.2.1
Hardware environment, 10.1
" independence, 10.4
Hashing, 9.2.2.1
Hold (data for programming user), 7.5.2.4
Host language capability, I.5.1, I.7.2
" " system, I.5.1, I.7.2, 1.1.2
Identifier, 2.2, 3.2
" ,entry, 3.5
" ,group, 2.2
" ,system, 2.2.3.3
Indexing (of stored data), 9.2.3
Index sequential storage structure, 9.2.3
Input mode, 7.2.1
" source file, 6.1, 6.2, 6.4
Instance, 2.0
" composition, 2.2.2.2, 2.3.2.2, 2.4.2.2, 2.5.2.2
" ,data base, 2.6.1
" ,entry, 2.4.2.2
" ,file, 2.5.2.2
" ,group, 2.2.2.2
" ,group relation, 2.3.2.2
" ,schema, 2.4.2.2
Integrity (of data), 7.4.5
" (of system), 5.4.4
Interactive mode (of operation), 10.3.2
Inter-entry, 2.5
Interfaces,programmer, 7.3, 7.6.1
" ,software, 10.2.4
Interrogation, I.8.4, 4.0
Inverted file, 9.2.3
Item, 2.1
Item attribute, maintenance of, 5.4.1
" ,count, 3.5
" identifier, 2.1.2.1
" instance, 2.1.3
" instance attribute, 4.7.2
" ,multiple-valued, 2.1.3

Item name, 2.1.2.1
  " number, 3.2
  " ,numeric, 2.1.1.1
  " ,principal, 2.2
  " schema, 2.1.2
  " " attribute, 2.1.2, 4.7.1
  " " definition, 3.2
  " storage, 9.1.1, 9.1.2
  " ,string, 2.1.1.2
  " type, 2.1.1
  " value class attribute, 4.7.3
Language type, 1.4
Level number, 3.3
Linked file, 2.5
Locate (data), 7.5.2.1
Locate & access (data), 7.5.2.2
Lock, access, 2.1
  " ,privacy, 3.2, 8.2.1
Logging (of modifications to the data base), 8.2.2
  " (of transactions), 8.2.3
  " records, 9.2.1
Logical connectors, 4.3.2.1
Mapping, data, 5.3.2
Mechanized file, 4.13
Modes of operation, 10.3
Modes of processing (data), 7.2
Modes of system use, 10.2.3
Modification statements (for programming user), 7.5.3
Monitor (during creation process), 6.1, 6.6
Multiple-valued item, 2.1.3
Multiprocessing, 10.2.1
Multiprogramming, 10.2.1
Name,entry, 2.4.3.1
  " ,file, 2.5.3.1
  " ,group, 2.2.3.1
  " ,group relation, 2.3.3.1
Network structures, I.6.3
Non-procedural capabilities, I.5.2, 4.1.2
Non-programming user, I.5.6, I.6.4, 7.1
Non-repeating group, 2.2, 9.1.2
Open (portion of data base), 7.5.1.1
Operational environment, I.8.10, 10.0
Operating system, 10.2.4.1
Ordering (of transactions and data files), 5.4.3
Output mode, 7.2.1
Parent group, 2.3
Pass keys, 8.2.1
Passwords, 8.2.1
Philosophy, underlying, 1.7.1
Physical records, 9.2.1
Plex entry, 2.4
Pointers (re: storage structure), 9.2.2.1

Population (of a file), 6.5
Premise action mix, 4.1
Prestored transactions, 5.0
Principal group, 2.2, 3.5
  "   item, 2.2
Privacy lock, 3.2, 8.2.1
Procedural capabilities, I.5.1, 4.1.1
Procedure modifications, 10.2.5
  "   preparation, 10.2.5
  "   submission, 10.2.5
Processor, 10.1.1
Programmer interface, 7.3, 7.6.1
Programming facilities, I.8.7, 7.0
  "   user, I.5.6, I.8.7, 7.0
Randomizing, 9.2.2.1
Random mode (of processing), 7.2.2
Records,logical, 9.2.1
  "   ,physical, 9.2.1
Reference quantity, 4.3.1.4
Relational conditions, 4.3.1.2, 4.3.1.4
  "   operators, 4.3.1.1, 4.3.1.3, 4.3.1.6
  "   storage structure, 9.2.2.2
Reorder (data) statements, 7.5.3.4
Reorganization (of data) statements, 7.5.3.5
Repeating group, 2.2
Report format facilities, 4.11
Restart, 10.1.2
Retrieval statements, 7.5.2
Scheduling,control over, 8.2.6
Schema, 2.0
  "   attribute, 2.0
  "   composition, 2.2.2.1, 2.3.2.1, 2.4.2.1, 2.5.2.1
  "   ,data base, 2.6.1
  "   definition,entry, 3.5
  "   "   ,file, 3.6
  "   "   ,group, 3.3
  "   "   ,group relation, 3.4
  "   ,entry, 2.4.2.1
  "   ,file, 2.5.2.1
  "   ,group, 2.2.2.1
  "   ,group relation, 2.3.2.1
  "   instance, 2.0
Security clearance (for programming user), 7.4.5
  "   (of data), 3.2, 7.4.5, 8.2
Selection criteria (for non-programming user), 4.0, 5.3.3
  "   "   (for programming user), 7.4.4
Self-contained capability, I.5.2, I.7.3
  "   system, I.5.2, I.7.3, 1.1.1
Separators, 9.1
Sequencer, 2.2, 3.2
Sequential mode (of processing), 7.2.2
  "   storage devices, 9.2.1
  "   "   structure, 9.2.2

Simple condition, 4.3.1
  "   group, 2.2
Software environment, 10.2
  "   interfaces, 10.2.4
Sort, 4.6.4
Space allocation, 6.3
Statements,change, 7.5.3.2
  "   ,close, 7.5.1.2
  "   ,control, 7.5.1
  "   ,delete, 7.5.3.3
  "   (for non-programming user), see Interrogation, Update
  "   (for programming user), see Programming Facilities
  "   ,hold, 7.5.3.4
  "   ,locate, 7.5.2.1
  "   ,locate & access, 7.5.2.2
  "   ,modification, 7.5.3
  "   ,open, 7.5.1.1
  "   ,reorder, 7.5.3.4
  "   ,reorganization, 7.5.3.5
  "   ,retrieval, 7.5.2
Storage devices, 9.2.1
  "   ,entry, 9.2, 9.3
  "   ,file, 9.2., 9.3
  "   ,item, 9.1.1, 9.1.2
  "   media, 9.2.1, 10.1.2
  "   (of programs), 8.2.5
  "   structure, I.6.1, I.8.9, 1.5, 2.0, 9.0
  "   "   ,chained, 9.2.2.2
  "   "   ,direct, 9.2.2.1
  "   "   (for creation), 6.2
  "   "   ,index sequential, 9.2.3
  "   "   ,sequential, 9.2.2
String item, 2.1.1.2
  "   matching conditions, 4.3.1.5
Subitem, 2.1.2.3, 3.2
System identifier, 2.2.3.3
Table handling, 7.5.4.1
Telecommunications statements, 7.5.4.2
Terminal equipment (support of), 10.1.3
Transaction, 5.0
  "   definition, 5.0, 5.2.1
  "   mode, 5.2.1, 10.3.3
  "   program, 5.0, 5.3
  "   "   definition, 5.3
Transferability, 10.4
Transformation (for creation), 6.5.2
  "   (of transactions), 5.3.7
Tree entry, 2.4
Type,entry, 2.4.1
  " ,file, 2.5.1
  " ,group, 2.2.1
  " ,group relation, 2.3.1

Unlinked file, 2.5
Update, I.8.5, 5.0
  " control (by user), 5.1
  " mode, 7.2.1
  " (with entry changes), 5.3.5.1
  " (with group changes), 5.3.5.2
  " (with item changes), 5.3.5.3
User working area, 7.4.2
Validation (for creation), 6.5.1
  " (of transactions), 5.3.6
Value class, 2.1.2.2
  " " attribute, 2.1.2.2, 4.7.3